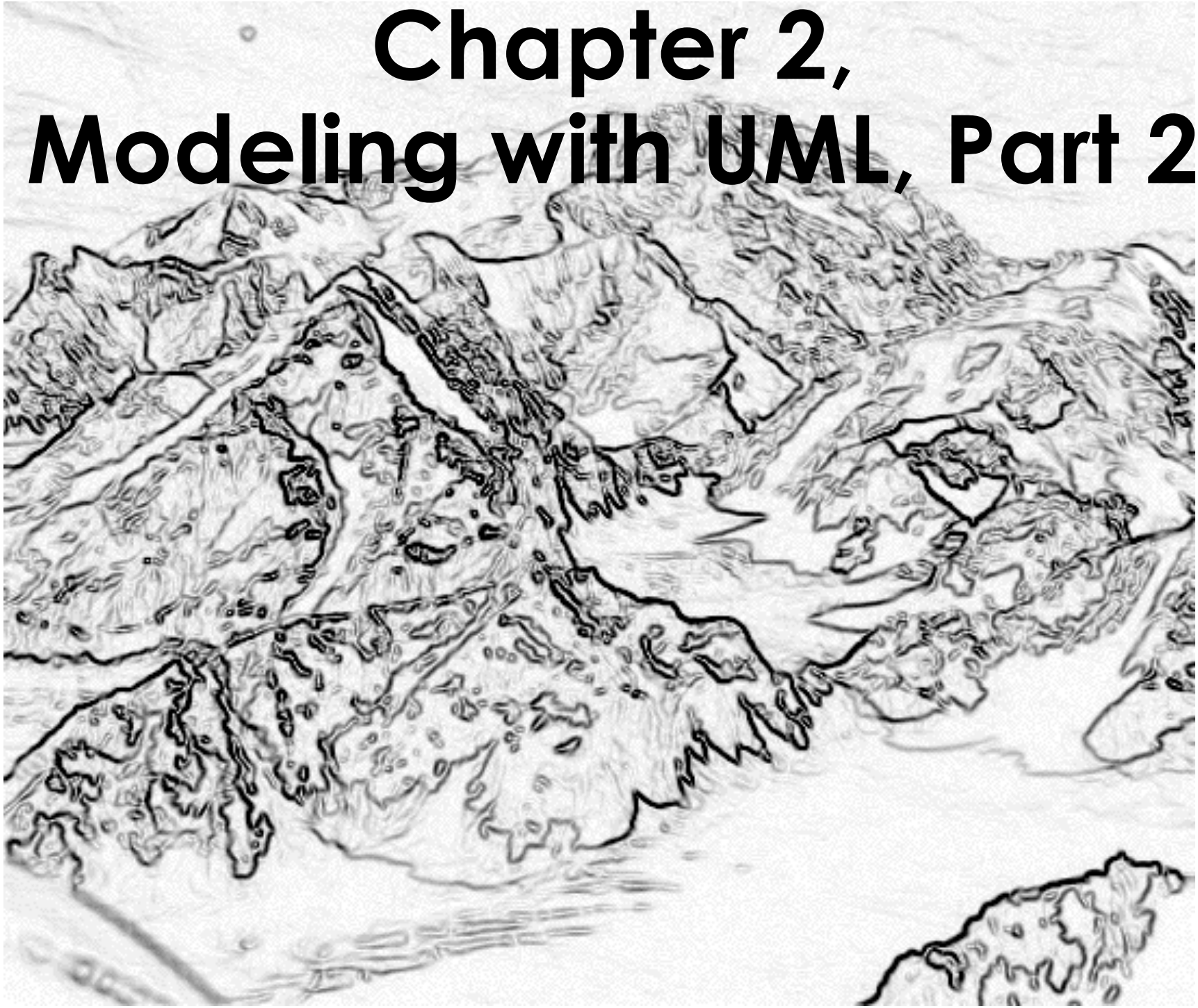
Object-Oriented Software Engineering
Using UML, Patterns, and Java

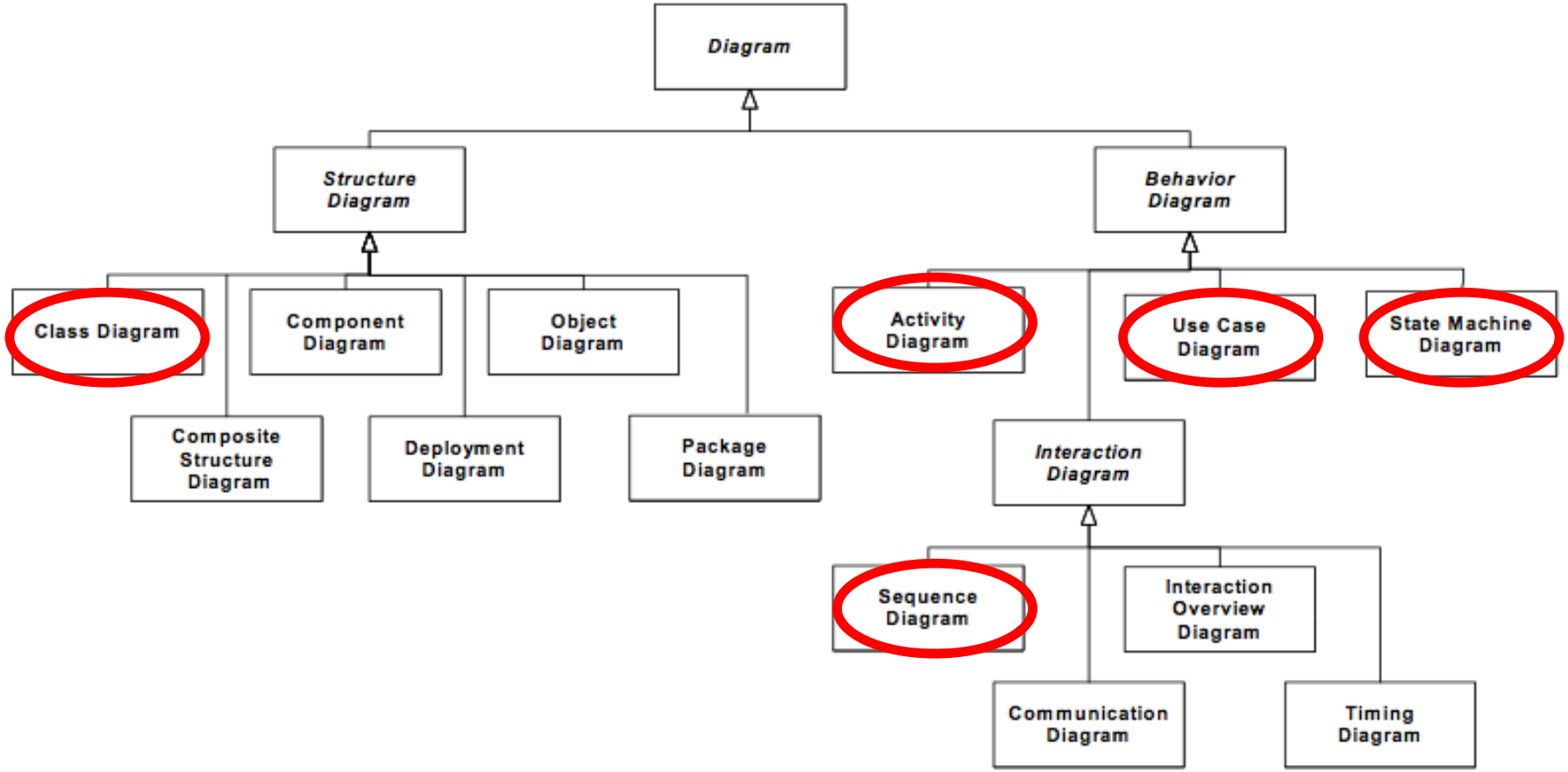
Chapter 2, Modeling with UML, Part 2



Outline of this Class

- Use case diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object
- Activity diagrams
  - Describe the dynamic behavior of a system, in particular the workflow.

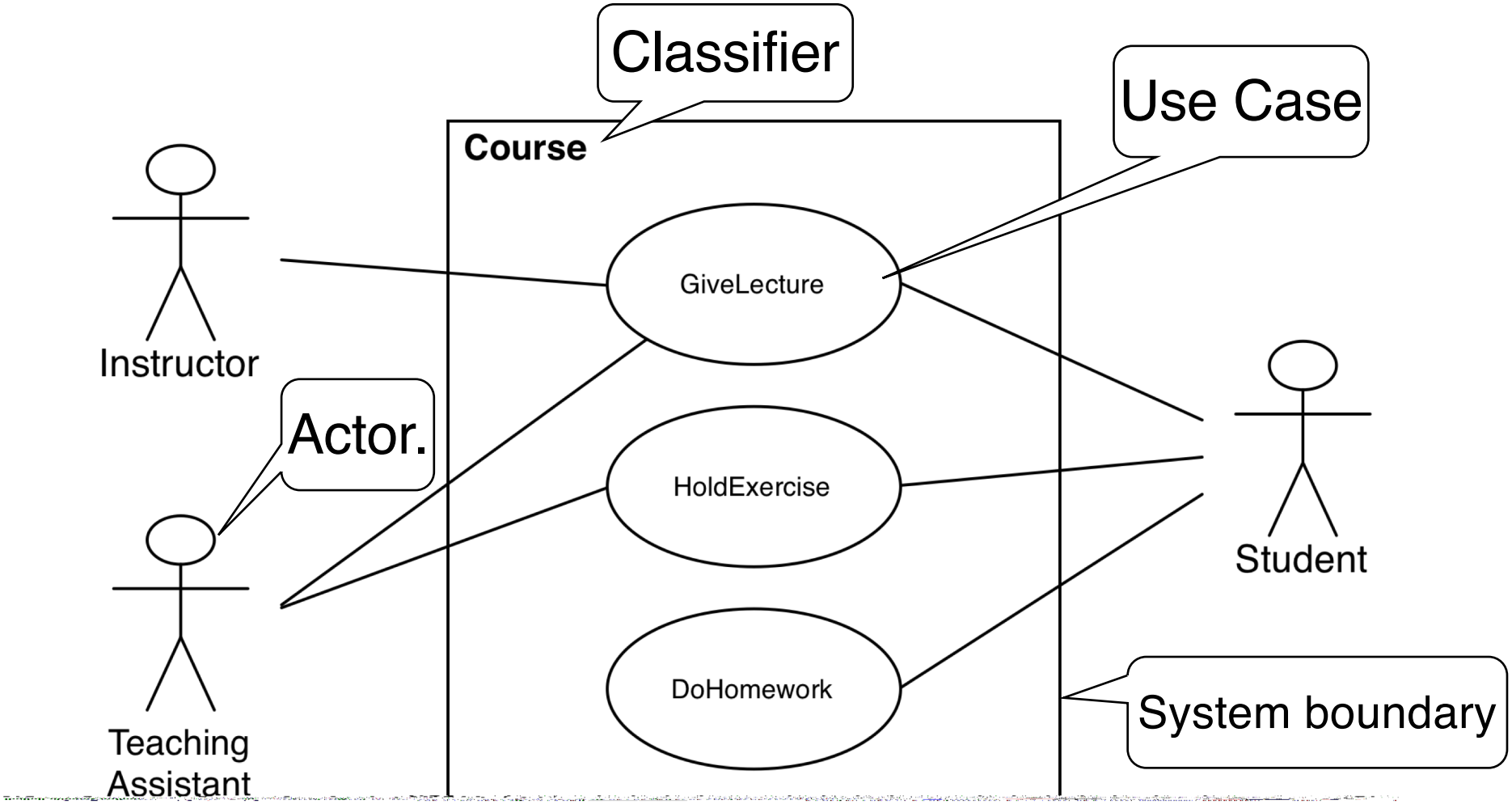
Another view on UML Diagrams



UML Basic Notation: First Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- In the first lecture we concentrated on:
  - Functional model: Use case diagram
  - Object model: Class diagram
  - Dynamic model: Sequence diagrams, statechart
- Now we go into a little bit more detail…

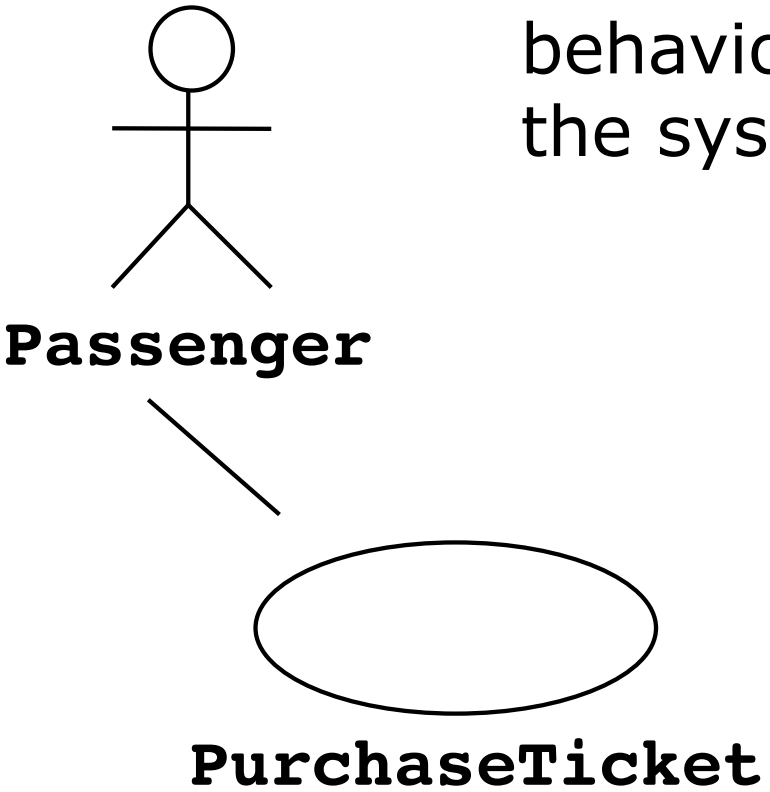
Use Case Model



Use case diagrams represent the functionality of the system from user's point of view

UML Use Case Diagrams

Used during requirements elicitation and analysis to represent external behavior ("visible from the outside of the system")



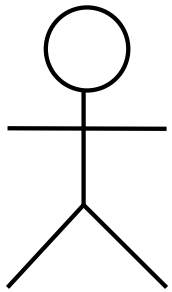
An ***Actor*** represents a role, that is, a type of user of the system

A ***use case*** represents a class of functionality provided by the system

Use case model:

The set of all use cases that completely describe the functionality of the system.

Actors



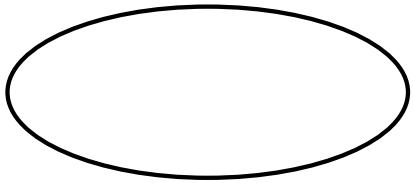
Passenger

- An actor is a model for an external entity which interacts (communicates) with the system:
  - User
  - External system (Another system)
  - Physical environment (e.g. Weather)
- An actor has a unique name and an optional description
- Examples:
  - **Passenger**: A person in the train
  - **GPS satellite**: An external system that provides the system with GPS coordinates.

Optional Description

Name

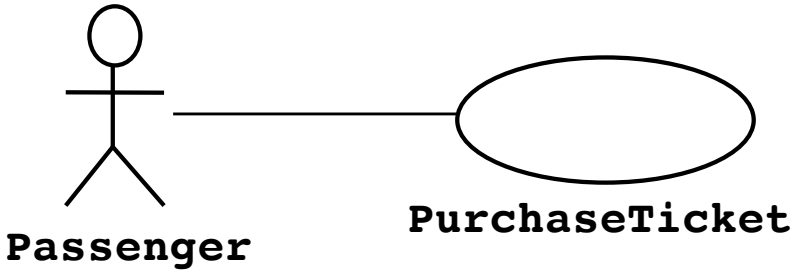
Use Case



PurchaseTicket

- A use case represents a class of functionality provided by the system
- Use cases can be described textually, with a focus on the event flow between actor and system
- The textual use case description consists of 6 parts:
  1. Unique name
  2. Participating actors
  3. Entry conditions
  4. Exit conditions
  5. Flow of events
  6. Special requirements.

Textual Use Case Description Example



1. Name: `Purchase ticket`

2. Participating actor:
`Passenger`

3. Entry condition:

- `Passenger` stands in front of ticket distributor
- `Passenger` has sufficient money to purchase ticket

4. Exit condition:

- `Passenger` has ticket

5. Flow of events:

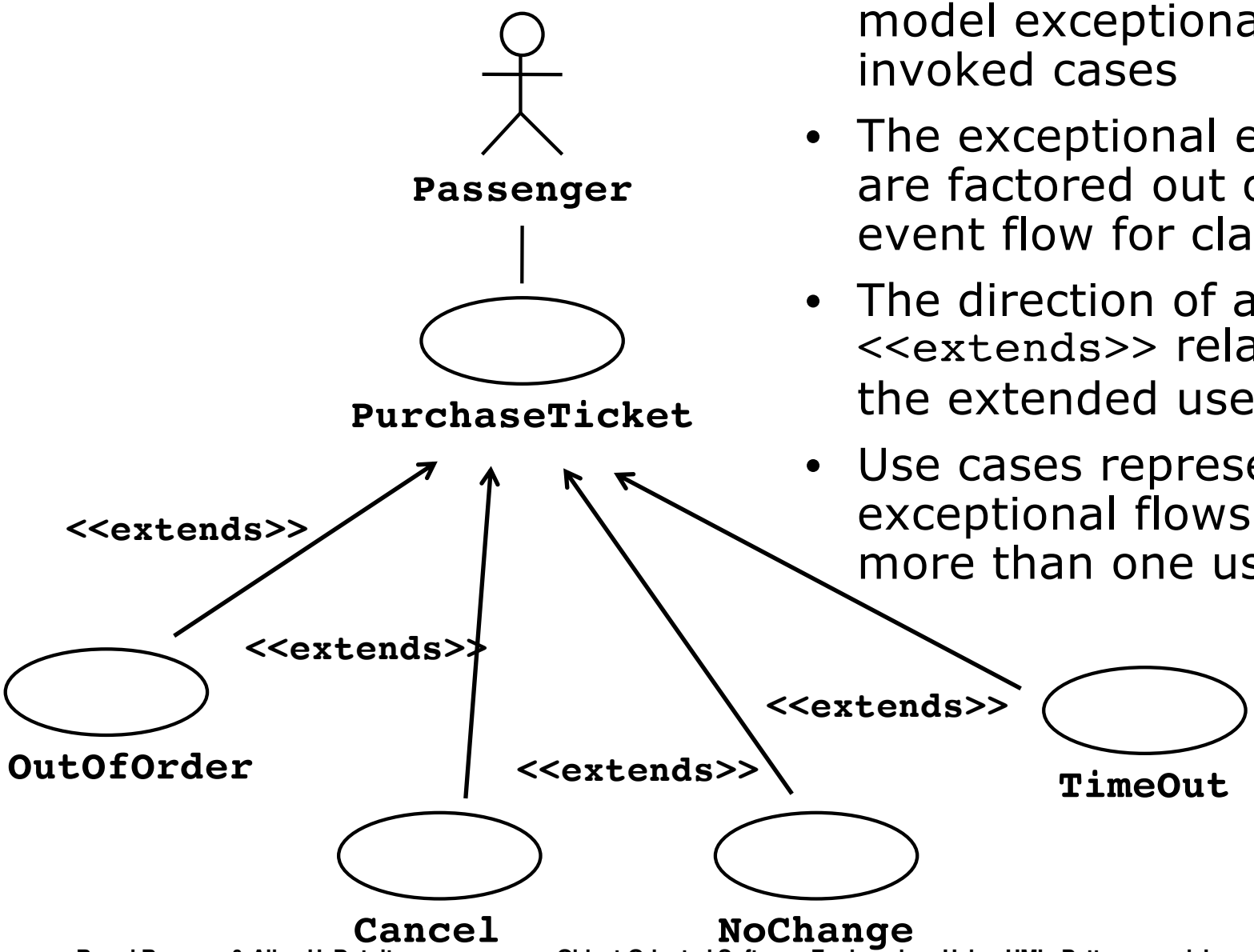
1. `Passenger` selects the number of zones to be traveled
2. Ticket Distributor displays the amount due
3. `Passenger` inserts money, at least the amount due
4. Ticket Distributor returns change
5. Ticket Distributor issues ticket

6. Special requirements:
None.

Uses Cases can be related

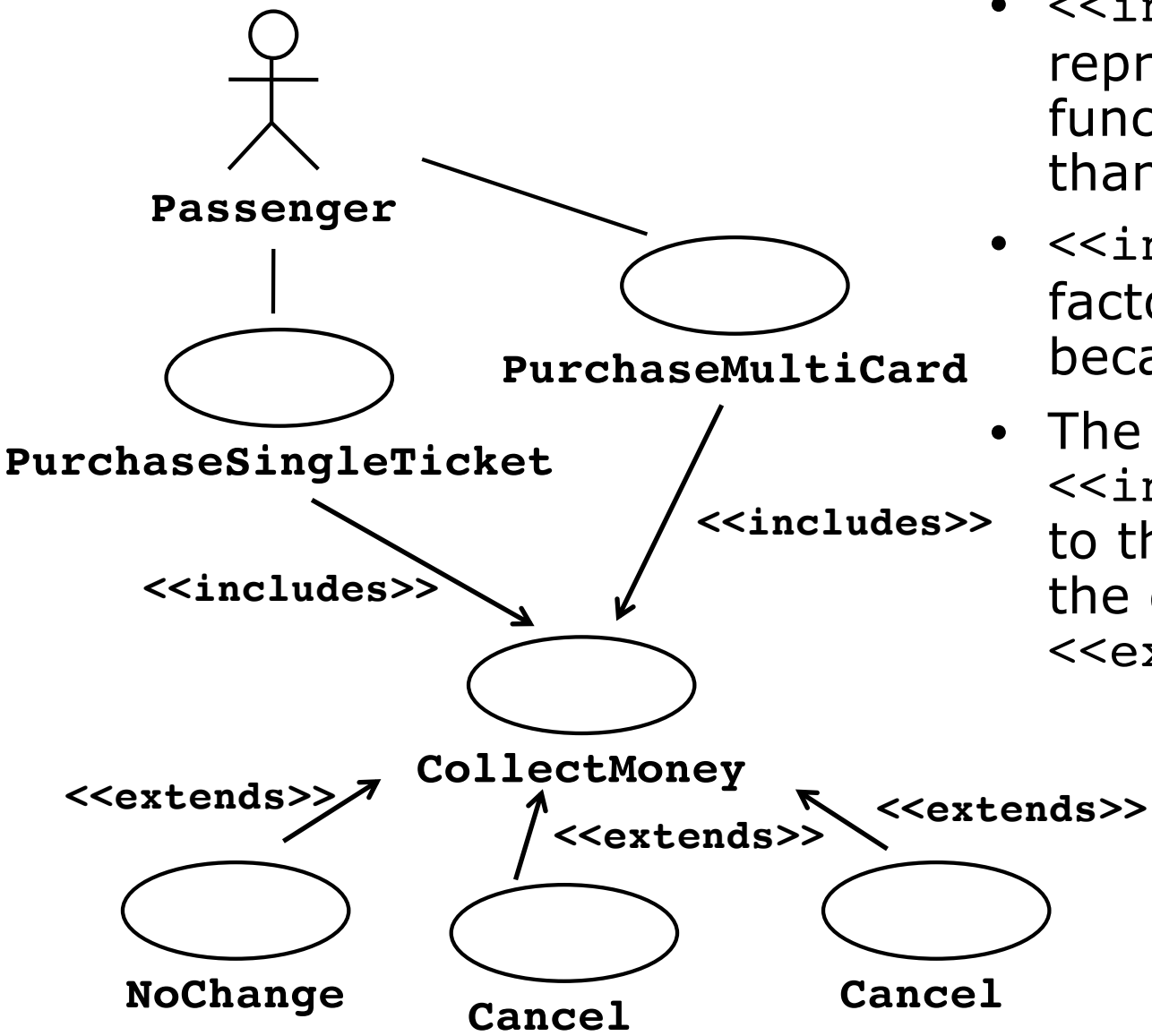
- Extends Relationship
  - To represent seldom invoked use cases or exceptional functionality
- Includes Relationship
  - To represent functional behavior common to more than one use case.

The <<*extends*>> Relationship



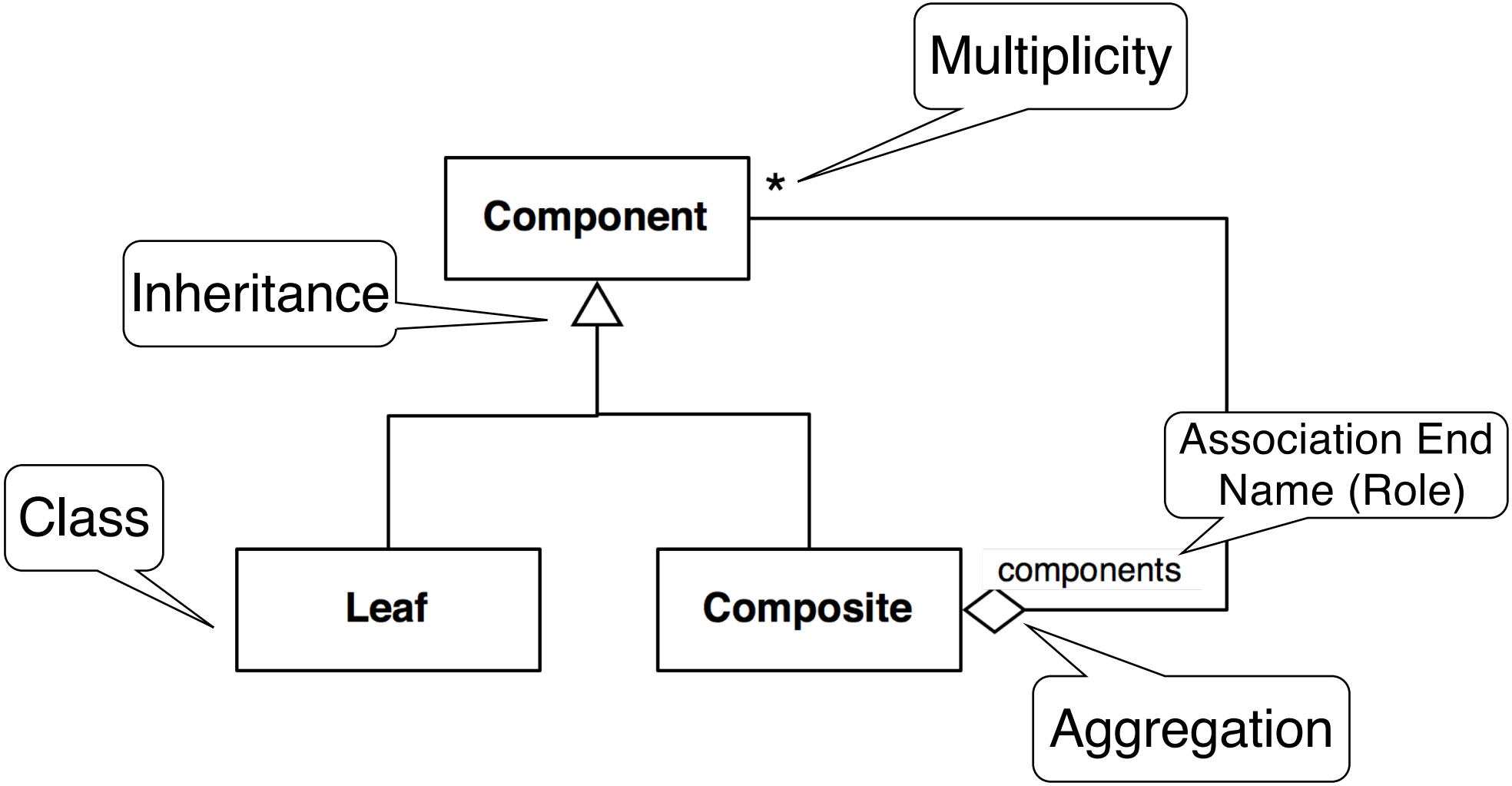
- `<<extends>>` relationships model exceptional or seldom invoked cases
- The exceptional event flows are factored out of the main event flow for clarity
- The direction of an `<<extends>>` relationship is to the extended use case
- Use cases representing exceptional flows can extend more than one use case.

The *<<includes>>* Relationship



- `<<includes>>` relationship represents common functionality needed in more than one use case
- `<<includes>>` behavior is factored out for reuse, not because it is an exception
- The direction of a `<<includes>>` relationship is to the using use case (unlike the direction of the `<<extends>>` relationship).

Review of Class Diagrams

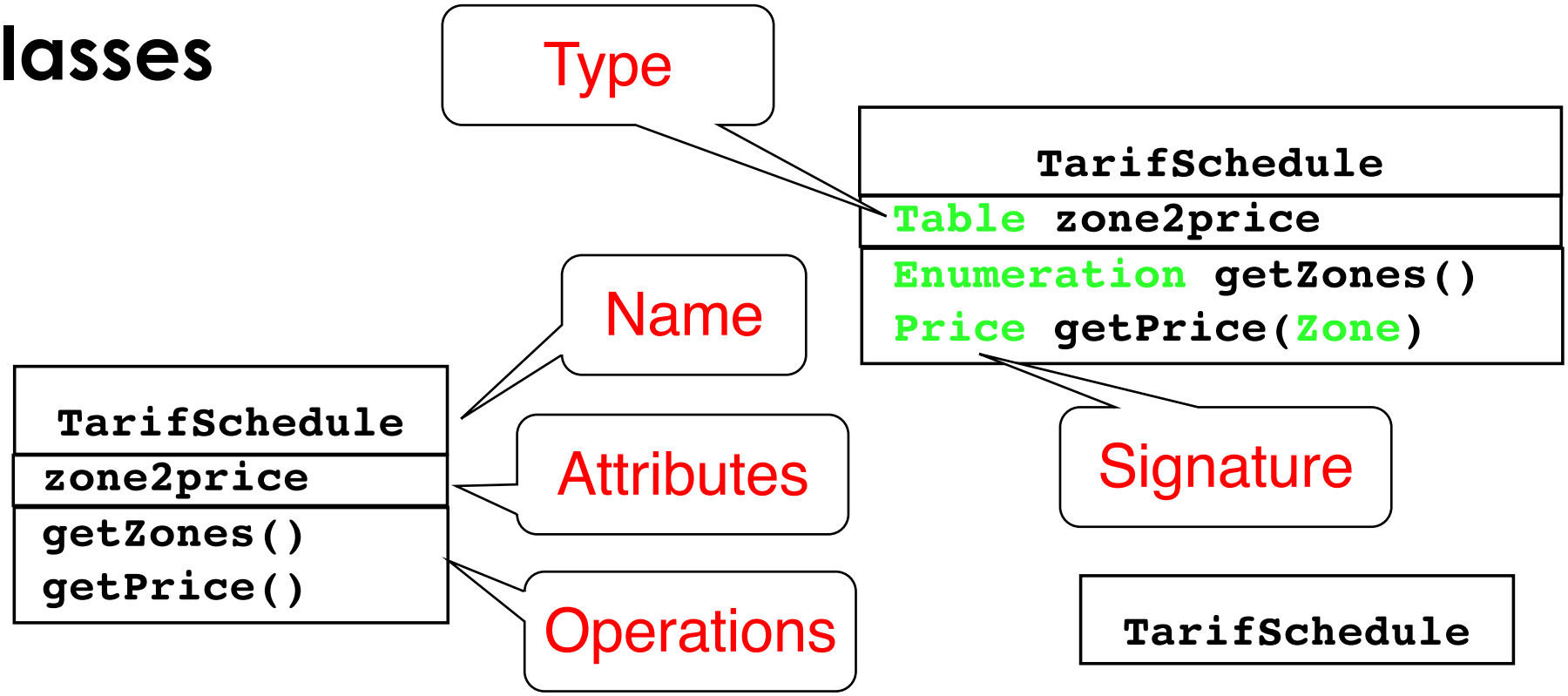


Class diagrams represent the structure of the system

Class Diagrams

- Class diagrams represent the structure of the system
- Used
  - during requirements analysis to model application domain concepts
  - during system design to model subsystems
  - during object design to specify the detailed behavior and attributes of classes.

Classes

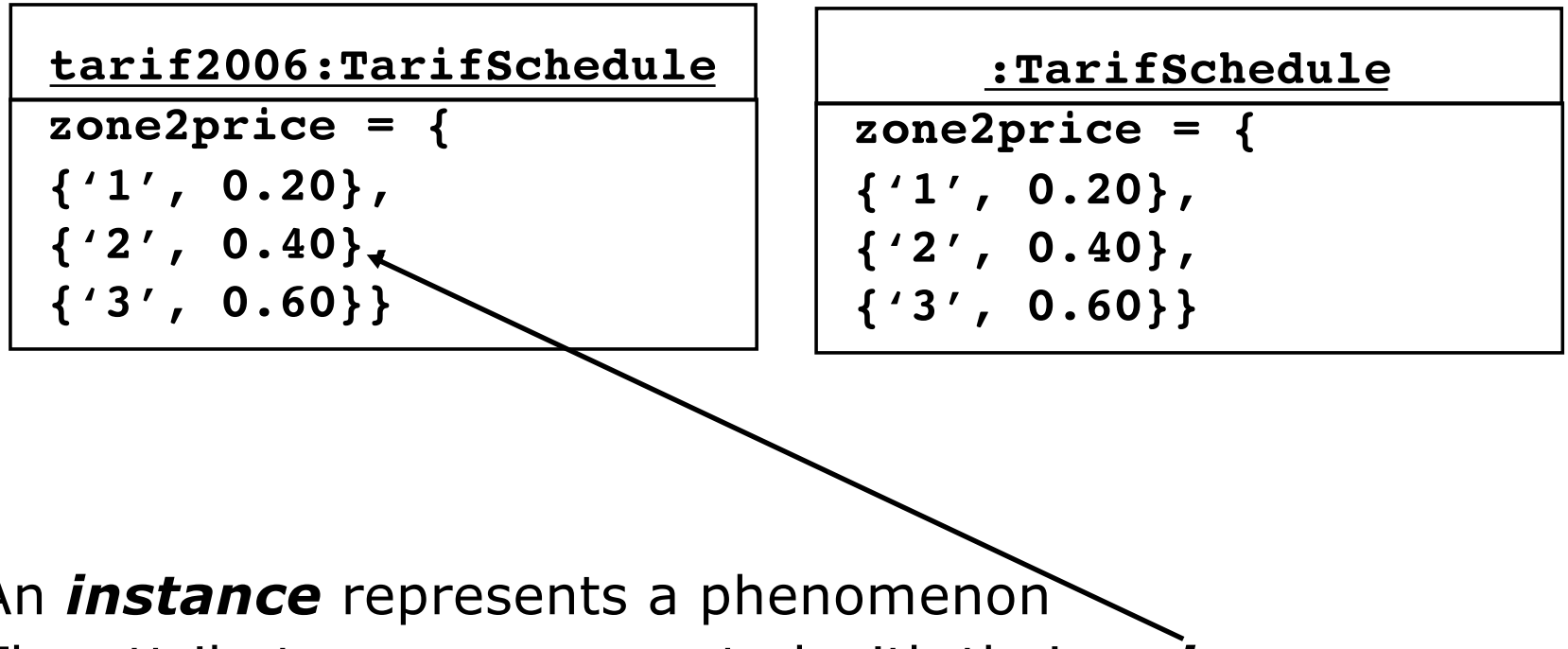


- A ***class*** represents a concept
- A class encapsulates state ***(attributes)*** and behavior ***(operations)***

  Each attribute has a ***type***
  Each operation has a ***signature***

The class name is the only mandatory information

Instances

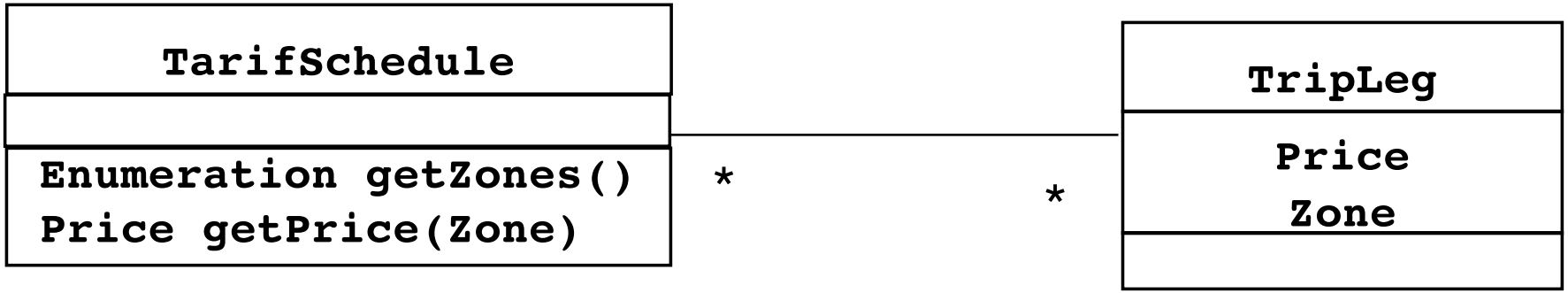


- An ***instance*** represents a phenomenon
- The attributes are represented with their ***values***
- The name of an instance is <u>underlined</u>
- The name can contain only the class name of the instance (anonymous instance)

Actor vs Class vs Object

- **Actor**
  - An entity outside the system to be modeled, interacting with the system ("Passenger")
- **Class**
  - An abstraction modeling an entity in the application or solution domain
  - The class is part of the system model ("User", "Ticket distributor", "Server")
- **Object**
  - A specific instance of a class ("Joe, the passenger who is purchasing a ticket from the ticket distributor").

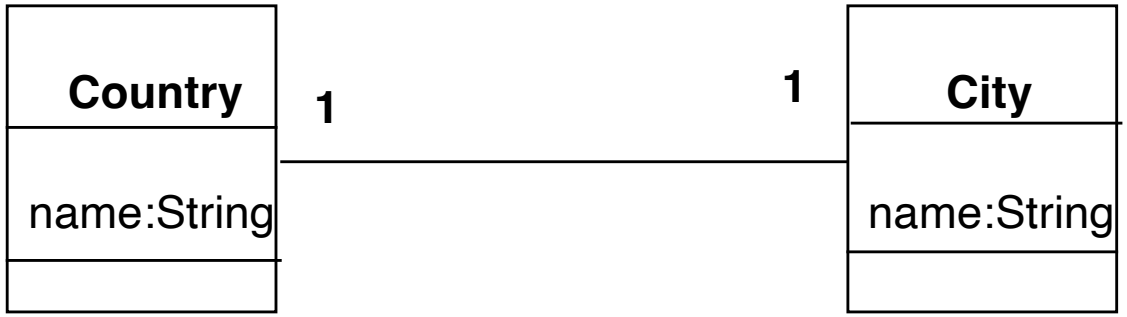
Associations



Associations denote relationships between classes

The multiplicity of an association end denotes how many objects the instance of a class can legitimately reference.

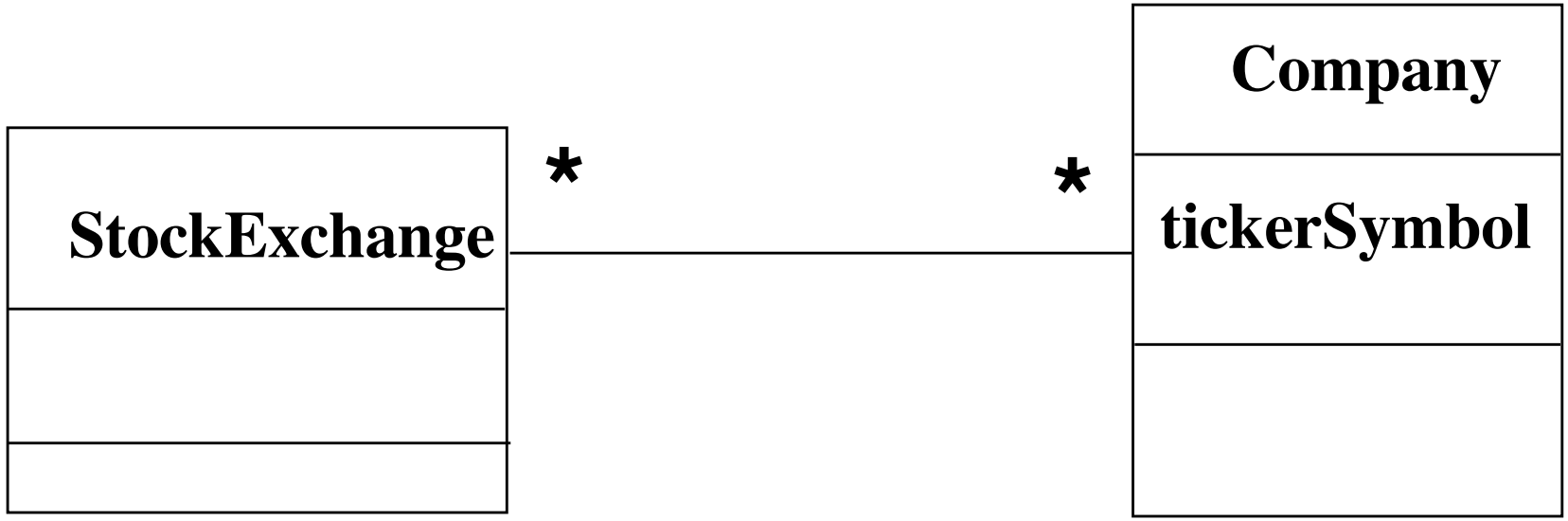
1-to-1 and 1-to-many Associations



1-to-1 association

1-to-many association

Many-to-Many Associations



From Problem Statement To Object Model

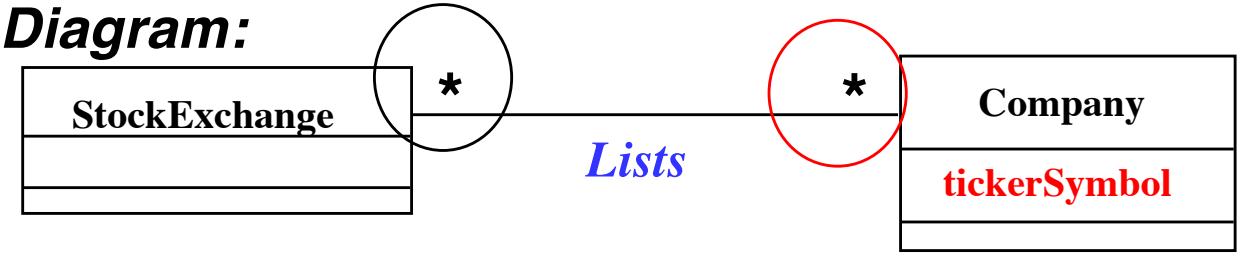
Problem Statement: A stock exchange lists many companies. Each company is uniquely identified by a ticker symbol

Class Diagram:

From Problem Statement to Code

Problem Statement : A stock exchange lists many companies. Each company is identified by a ticker symbol

Class Diagram:



Java Code

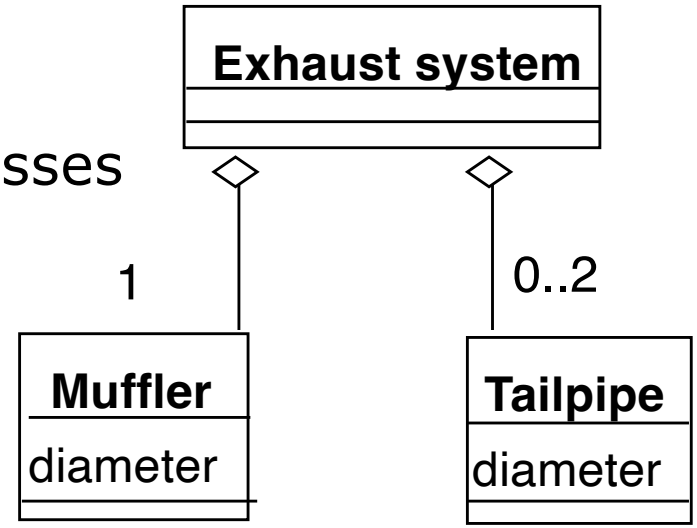
```
public class StockExchange
{
 private Vector m_Company = new Vector();
};

public class Company
{
 public int m_tickerSymbol;
 private Vector m_StockExchange = new Vector();
};
```

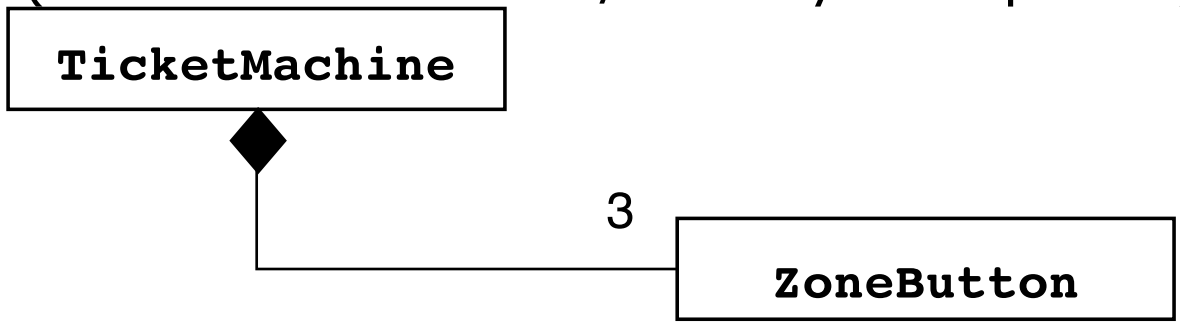
Associations are mapped to Attributes!

Aggregation

- An *aggregation* is a special case of association denoting a "consists-of" hierarchy
- The *aggregate* is the parent class, the components are the children classes

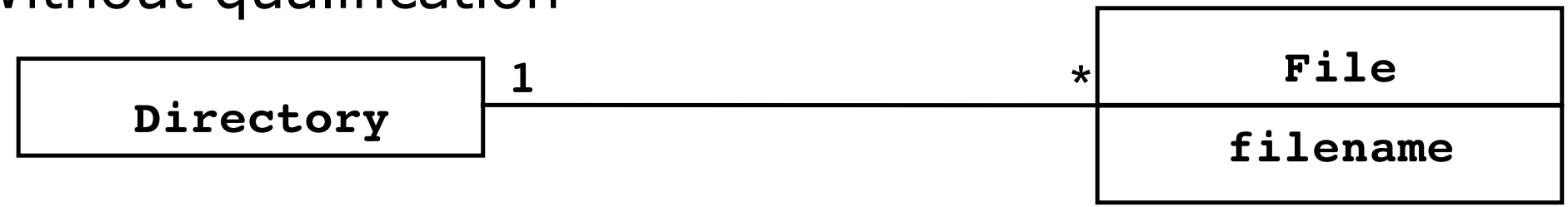


A solid diamond denotes *composition*: A strong form of aggregation where the *life time of the component instances* is controlled by the aggregate. That is, the parts don't exist on their won ("the whole controls/destroys the parts")



Qualifiers

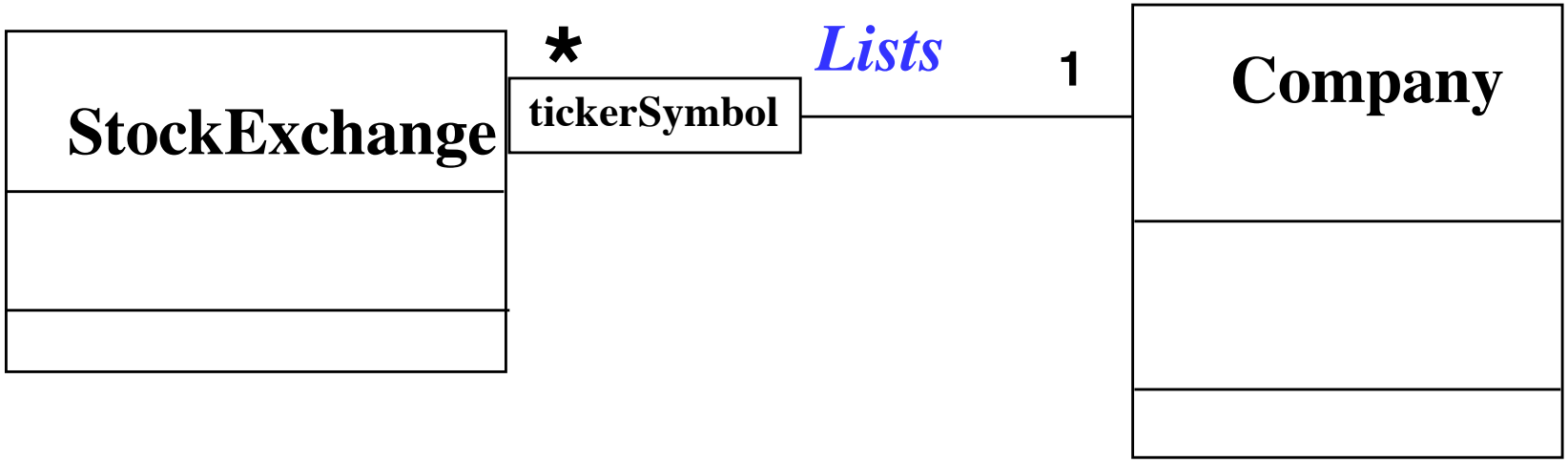
Without qualification



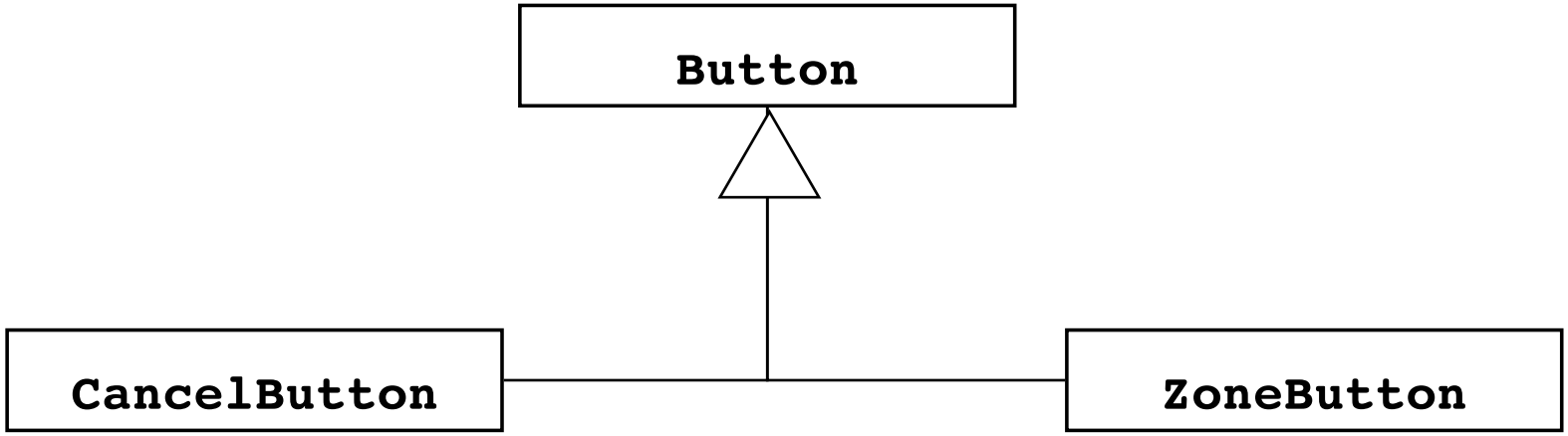
With qualification

- Qualifiers can be used to reduce the multiplicity of an association

Qualification: Another Example



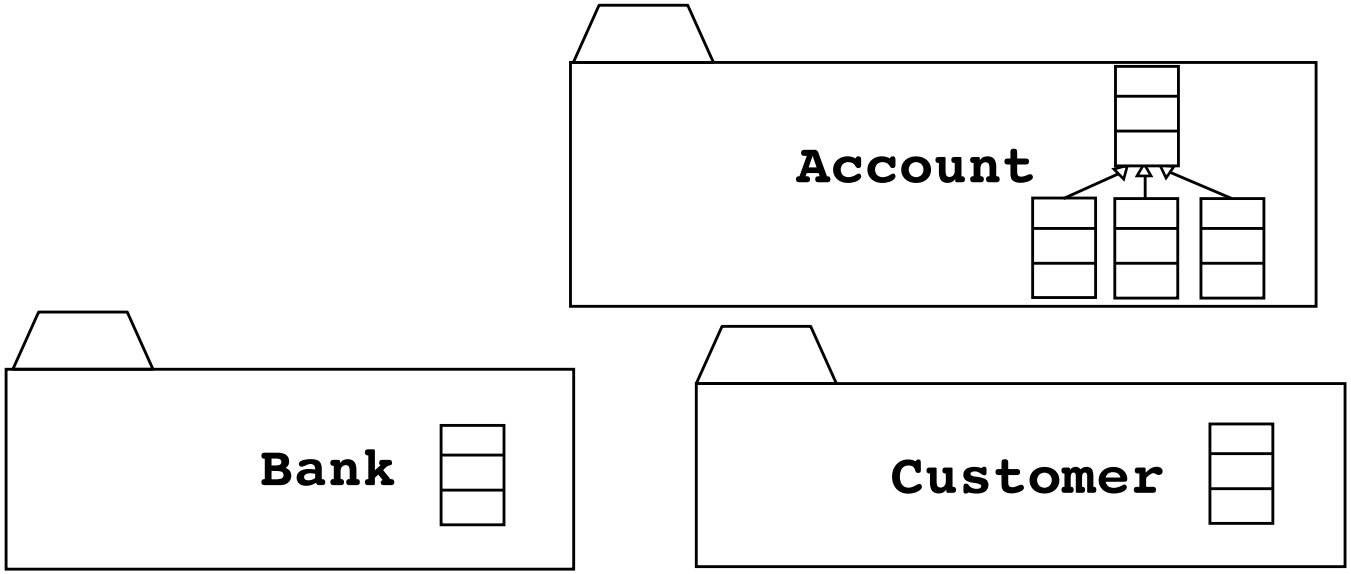
Inheritance



- *Inheritance* is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The **children classes** inherit the attributes and operations of the **parent class.**

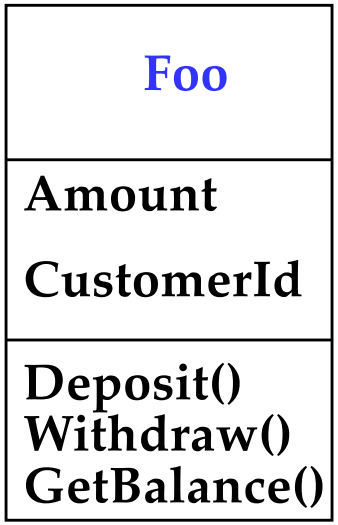
Packages

- Packages help you to organize UML models to increase their readability
- We can use the UML package mechanism to organize classes into subsystems



- Any complex system can be decomposed into subsystems, where each subsystem is modeled as a package.

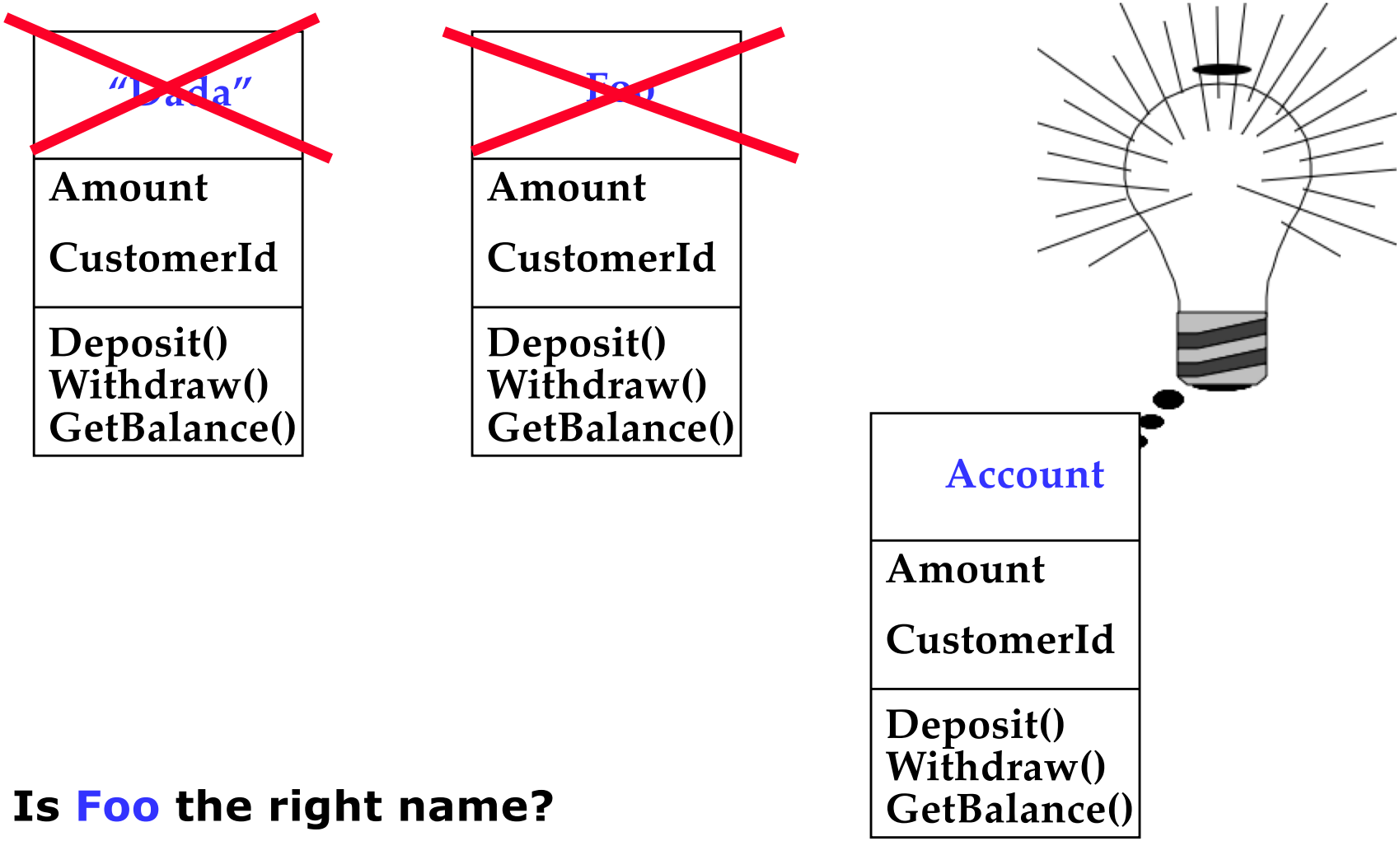
Object Modeling in Practice



Class Identification: Name of Class, Attributes and Methods

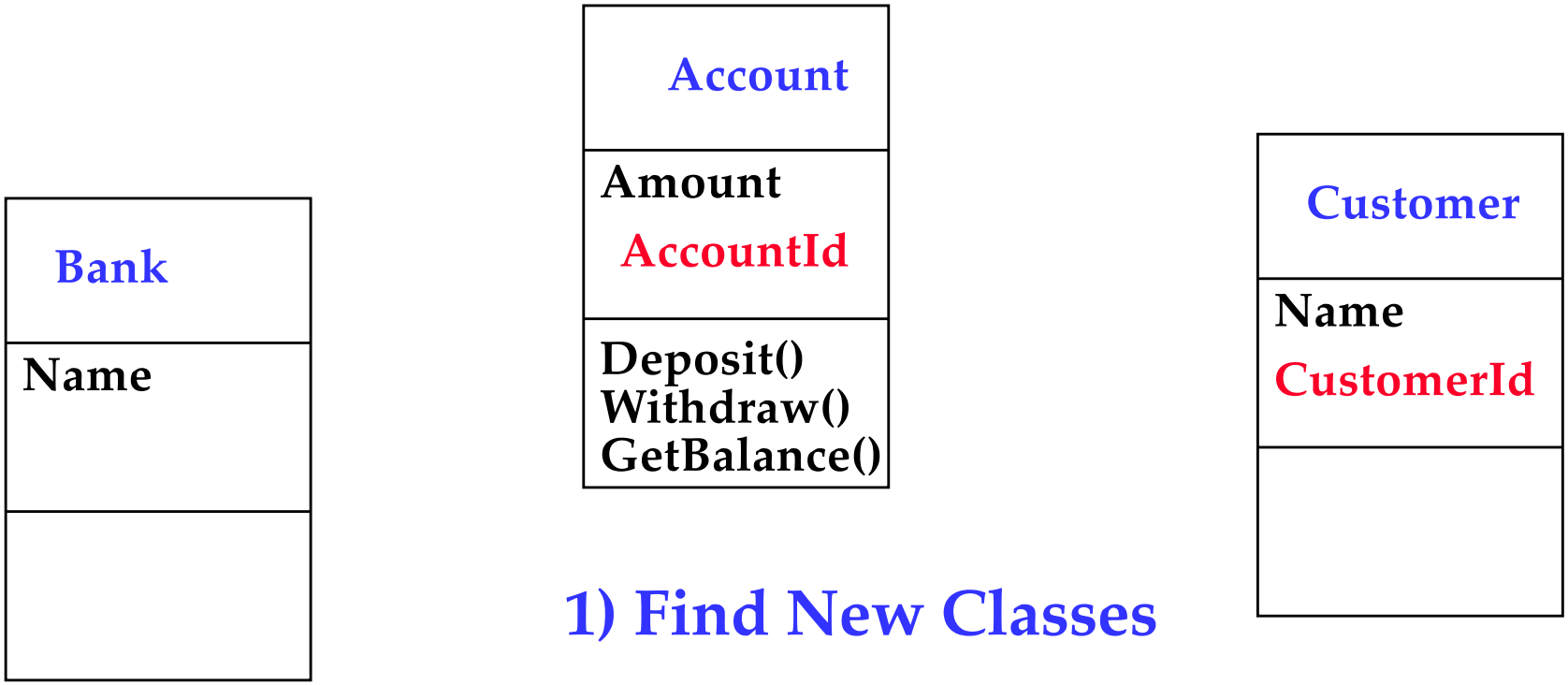
Is Foo the right name?

Object Modeling in Practice: Brainstorming



Is Foo the right name?

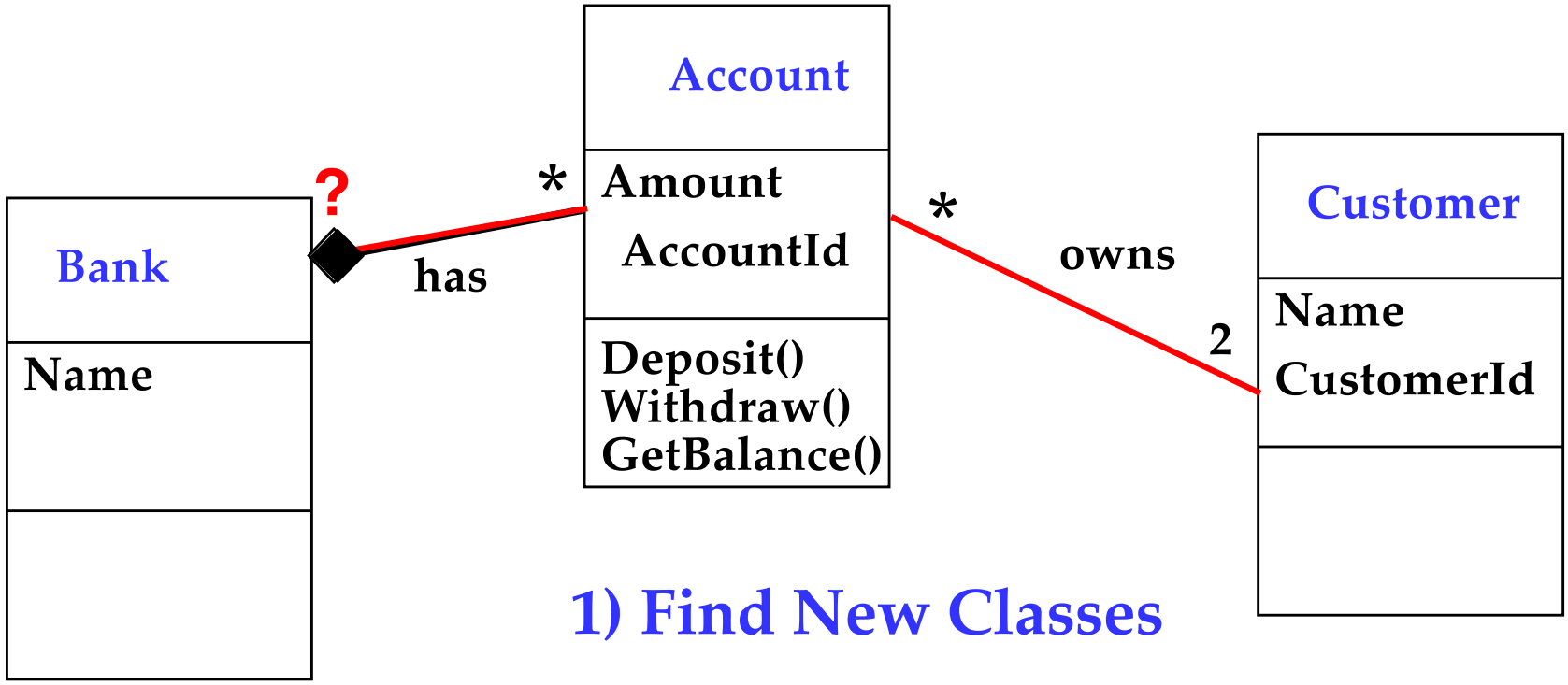
Object Modeling in Practice: More classes



1) Find New Classes

2) Review Names, Attributes and Methods

Object Modeling in Practice: Associations



1) Find New Classes

2) Review Names, Attributes and Methods

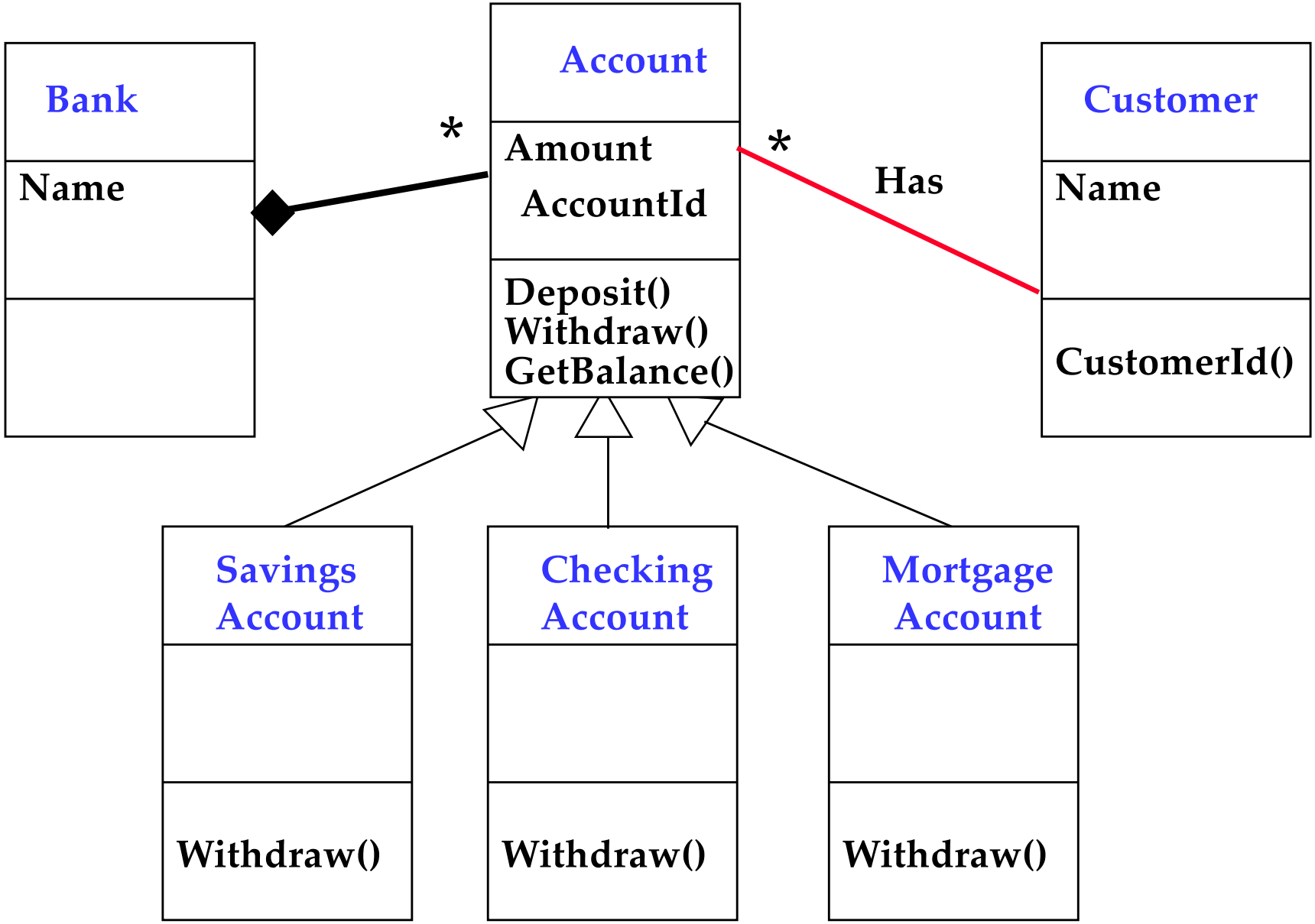
3) Find Associations between Classes

4) Label the generic assocations

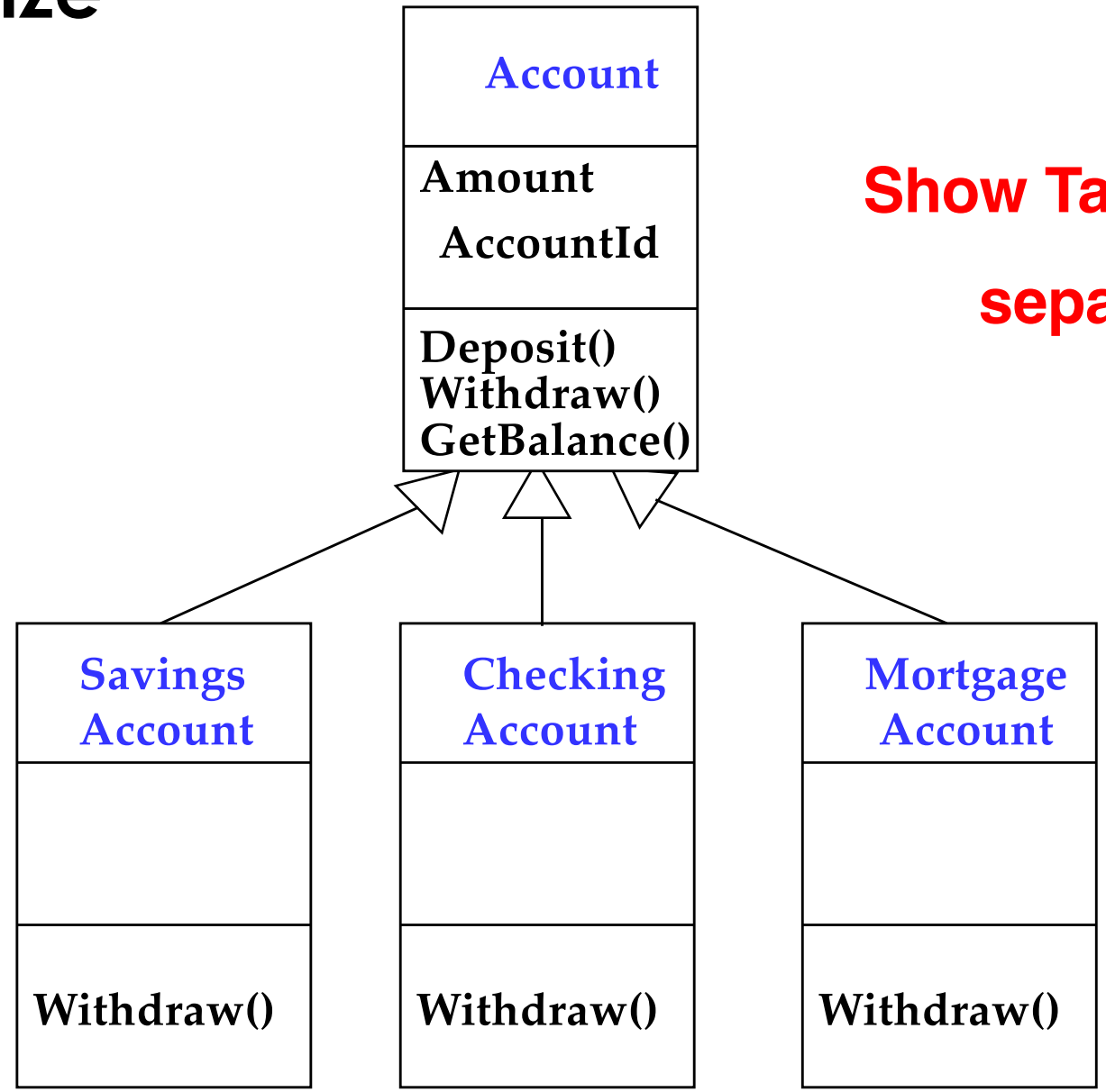
5) Determine the multiplicity of the assocations

6) Review associations

Practice Object Modeling: Find Taxonomies

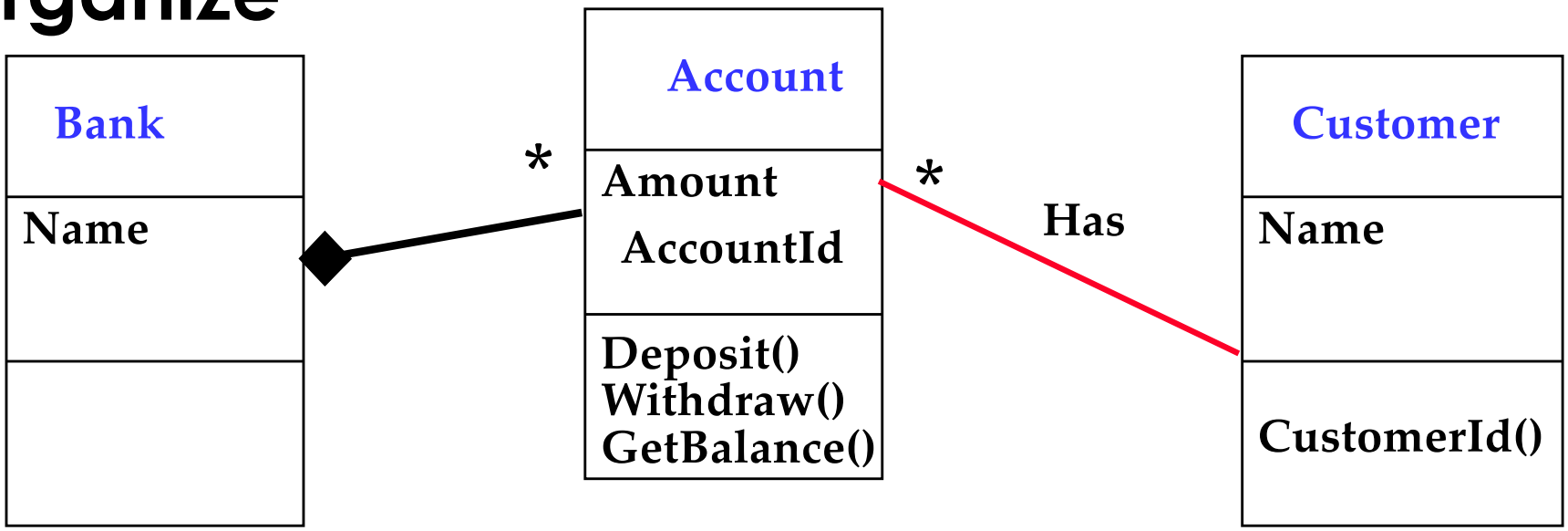


Practice Object Modeling: Simplify, Organize



Show Taxonomies separately

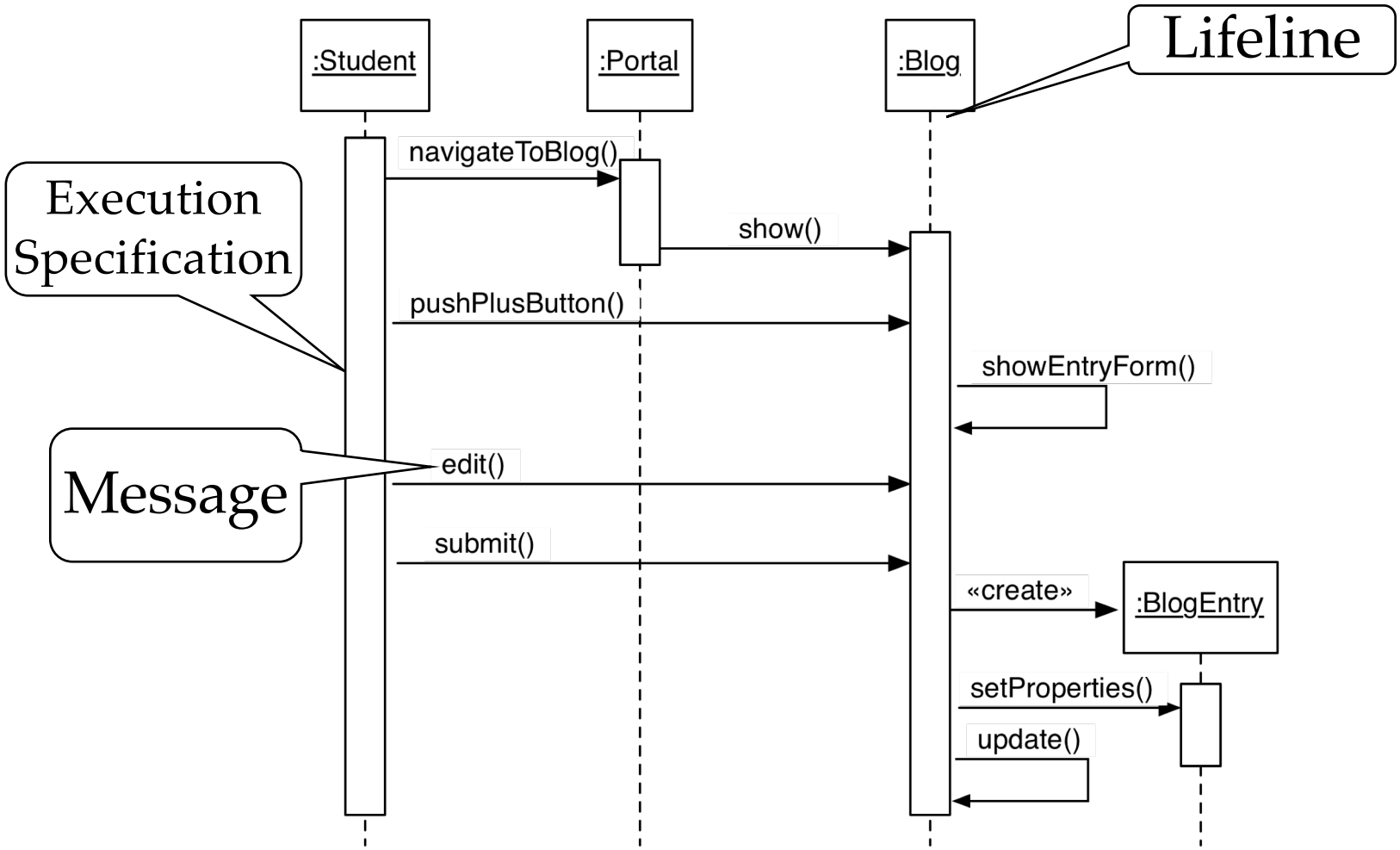
Practice Object Modeling: Simplify, Organize



Use the 7+-2 heuristics

or better 5+-2!

Sequence diagram: Basic Notation



Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*.

Lifeline and Execution Specification

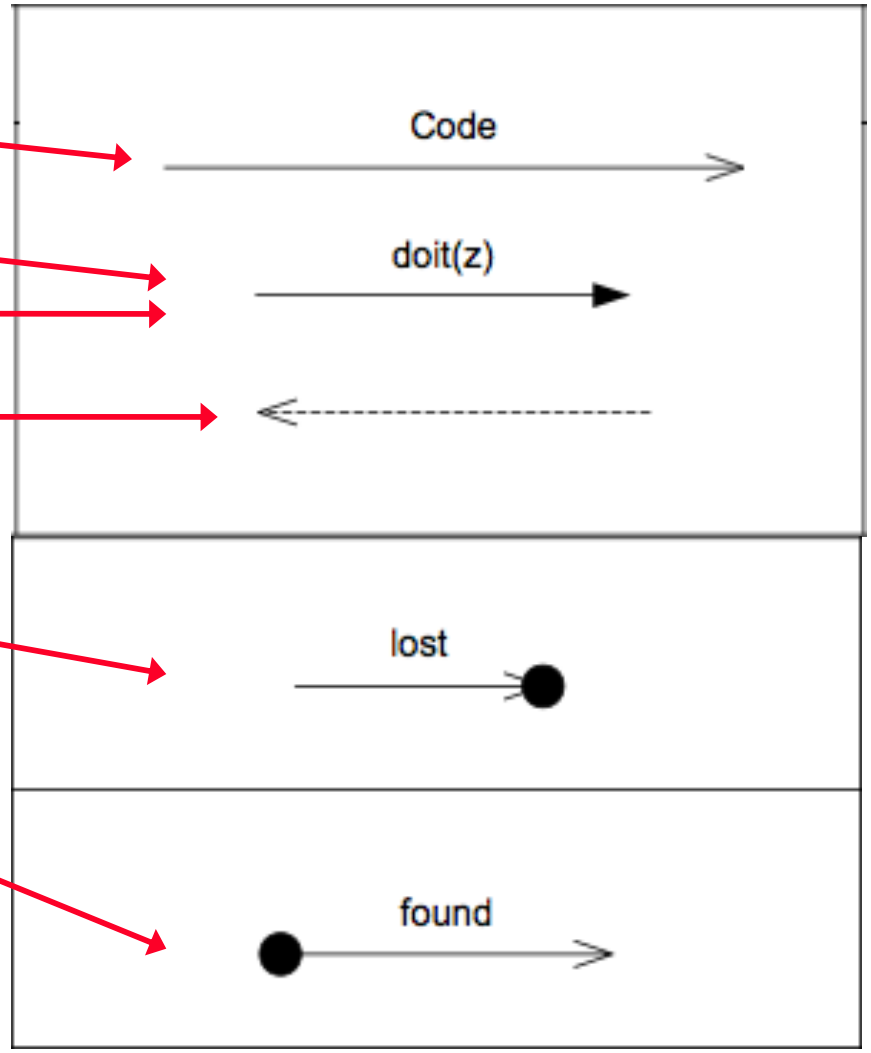
- A **lifeline** represents an individual participant (or object) in the interaction
- A lifeline is shown using a symbol that consists of a rectangle forming its “head” followed by a vertical line (which may be dashed) that represents the lifetime of the participant
- An **execution specification** specifies a behavior or interaction within the lifeline
- An execution specification is represented as a thin rectangle on the lifeline.

Messages

- Define a particular communication between lifelines of an interaction
- Examples of communication
  - raising a signal
  - invoking an operation
  - creating or destroying an instance
- Specify (implicitly) sender and receiver
- are shown as a line from the sender to the receiver
- Form of line and arrowhead reflect message properties

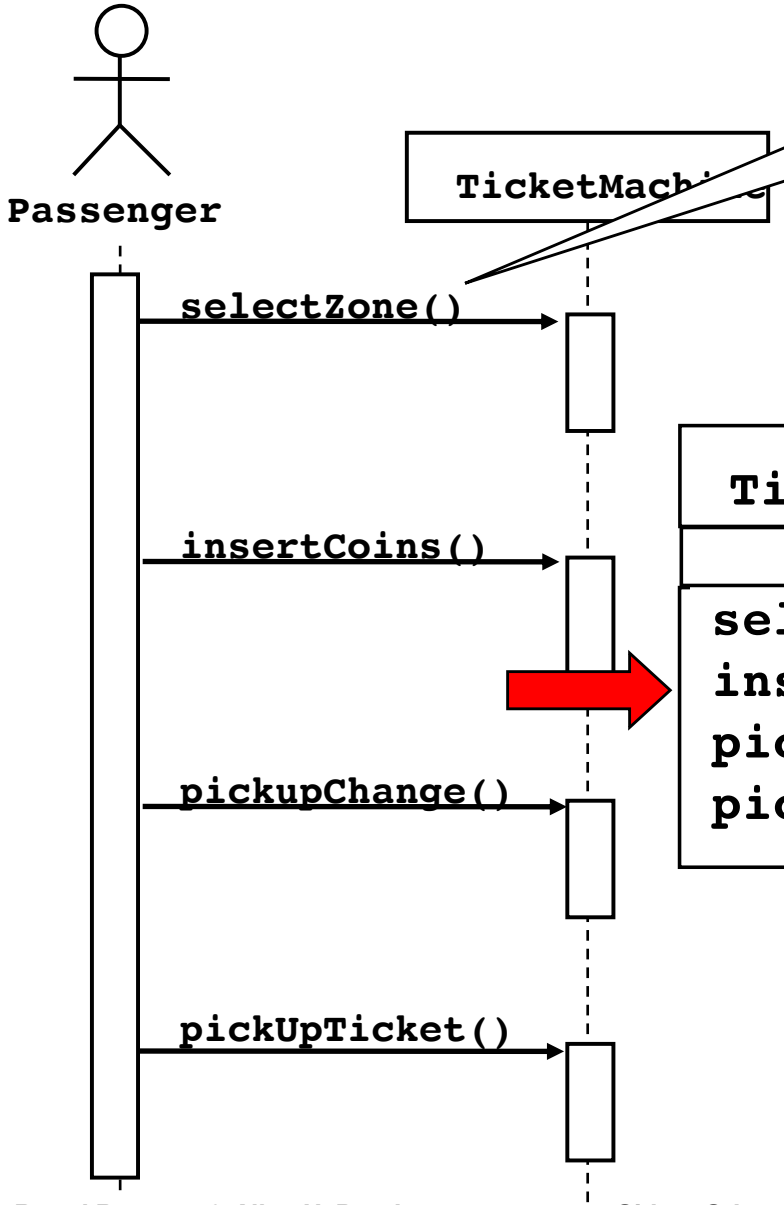
Message Types

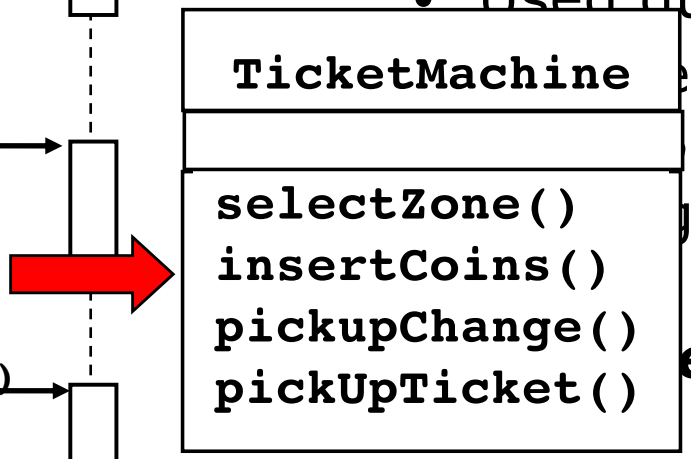
- Asynchronous
- Synchronous
- Call and Object creation
- Reply
- Lost
- Found



Sequence Diagrams

Focus on Controlflow





Used during analysis

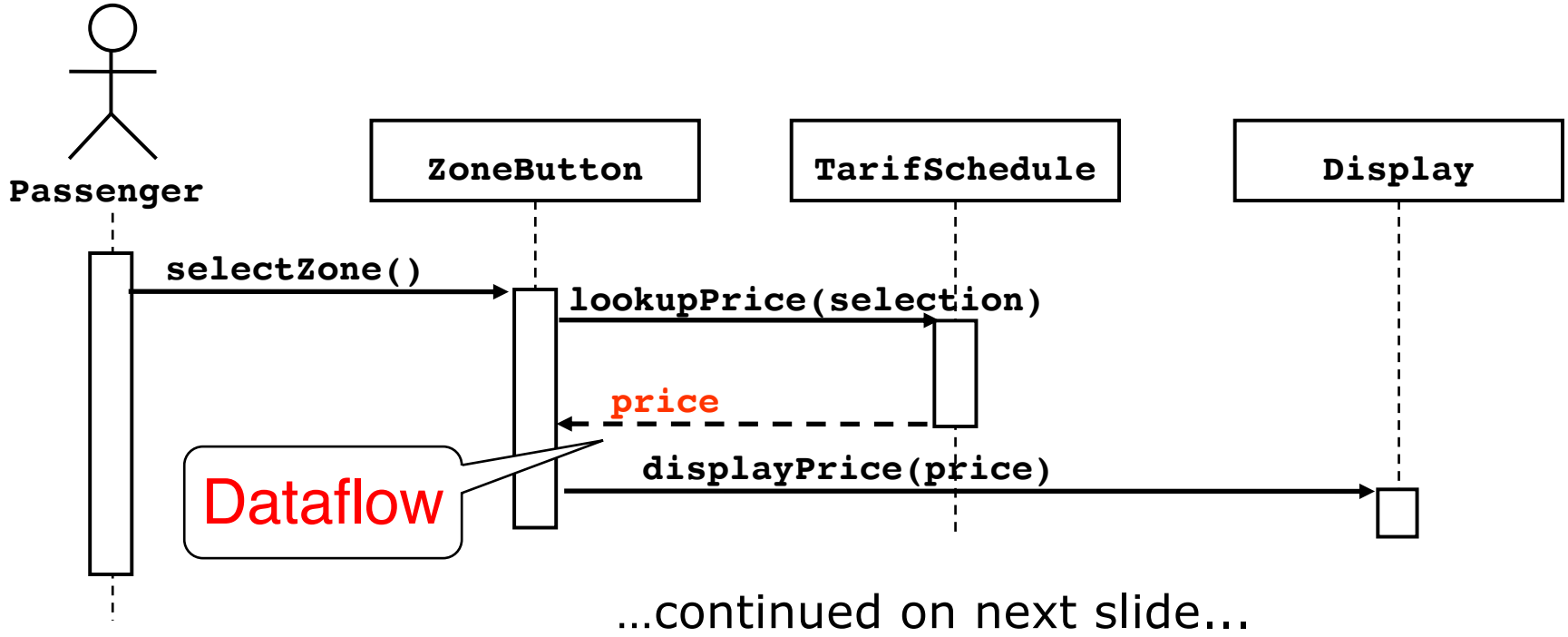
- To refine use case descriptions
- to find additional objects ("participating objects")

- Used during system design
  - efine subsystem interfaces
  - ces ar
  - les. Act

Messages -> Operations on participating Object

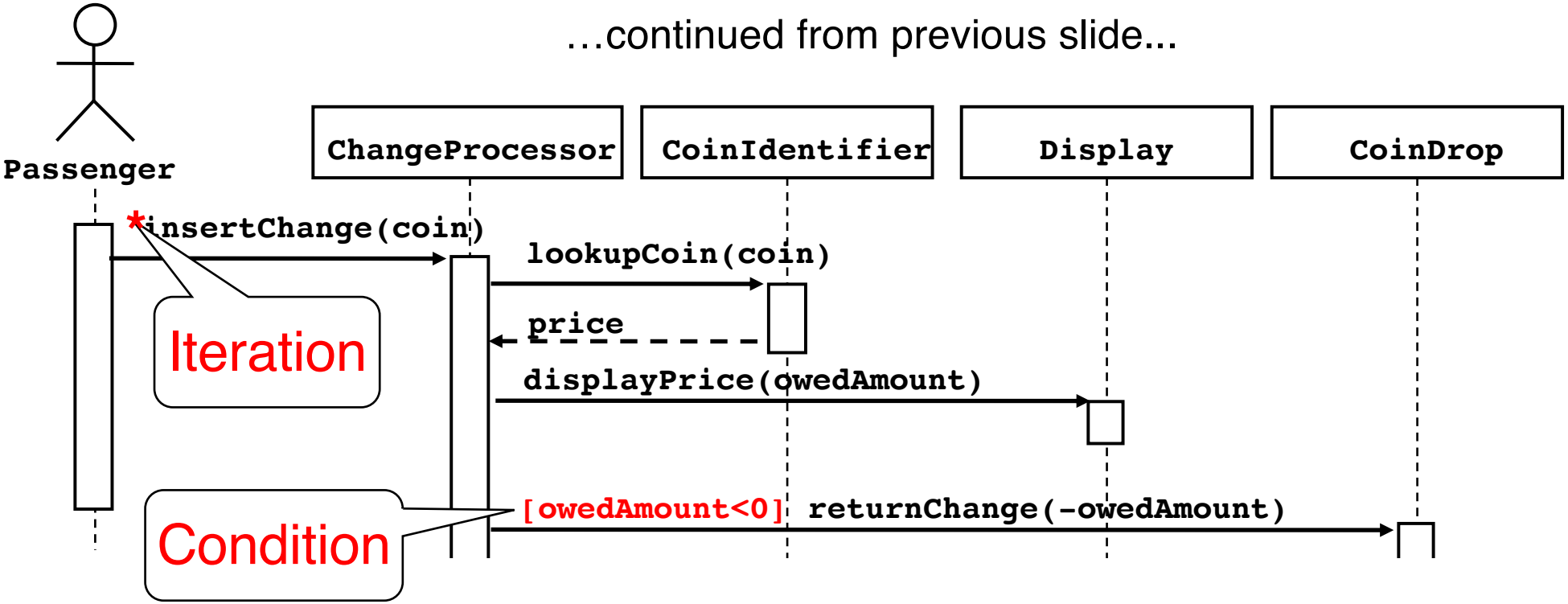
- es are represented by lines
- ***Messages*** are represented by arrows
- ***Activations*** are represented by narrow rectangles.

Sequence Diagrams can also model the Flow of Data



- The source of an arrow indicates the activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message

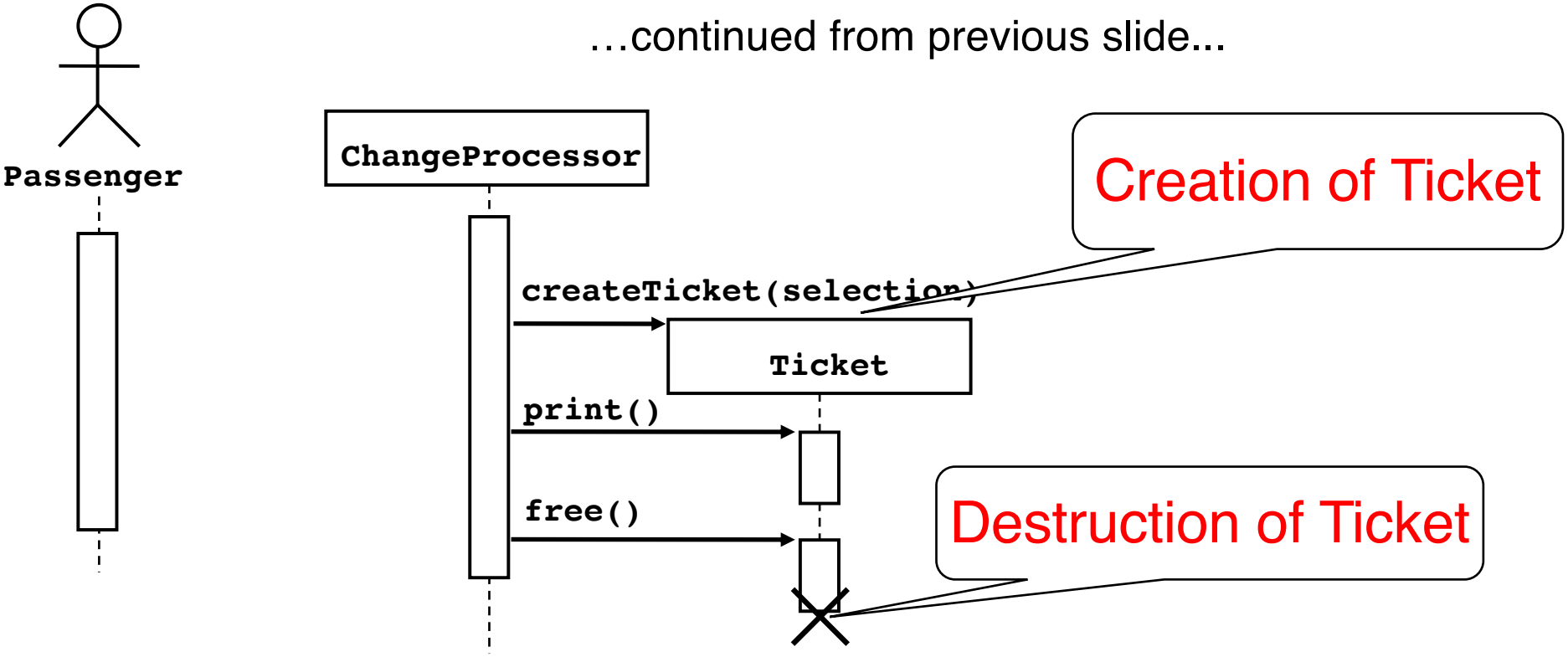
Sequence Diagrams: Iteration & Condition



…continued on next slide…

- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

Creation and destruction

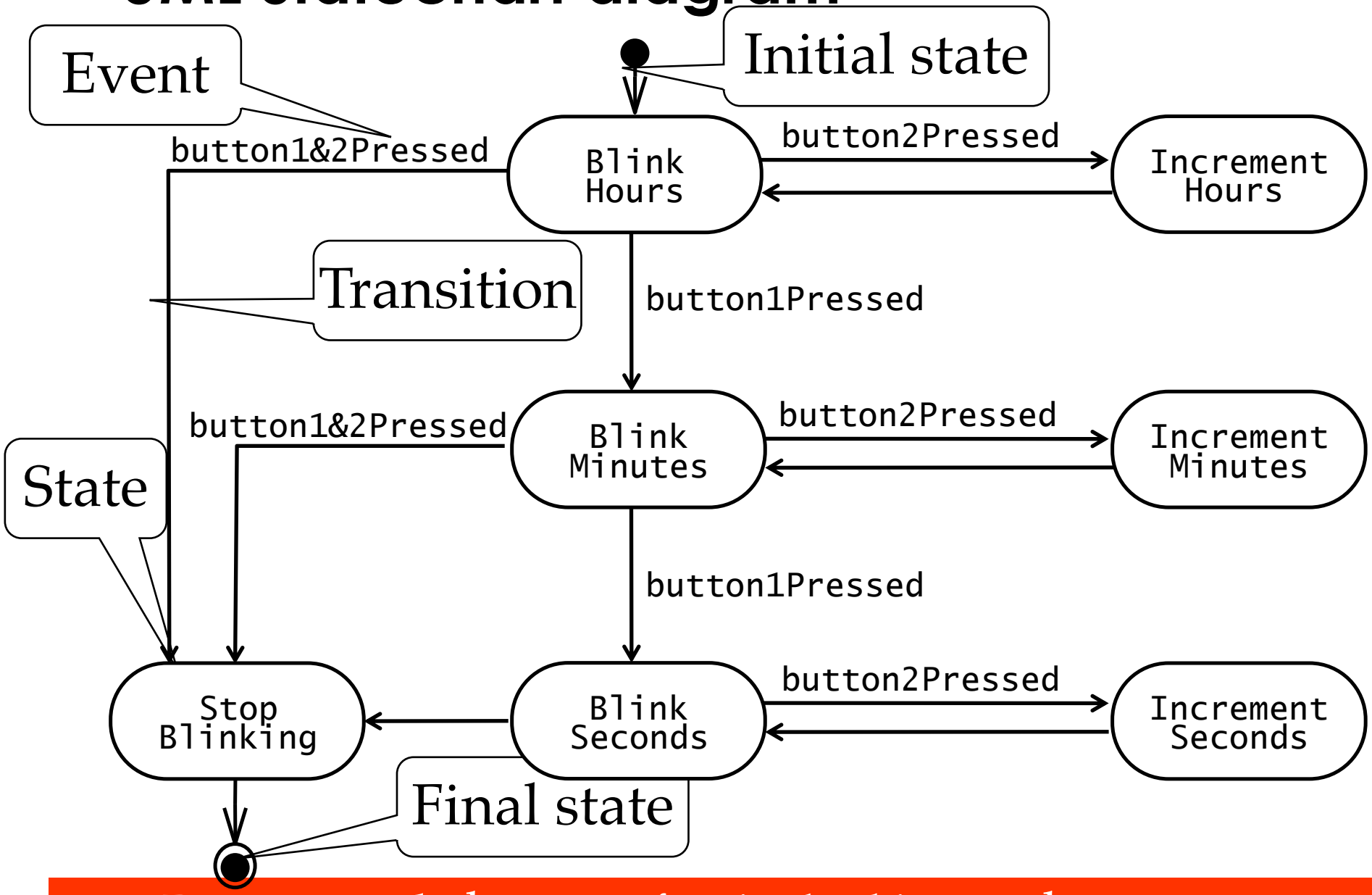


- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
  - In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

Sequence Diagram Properties

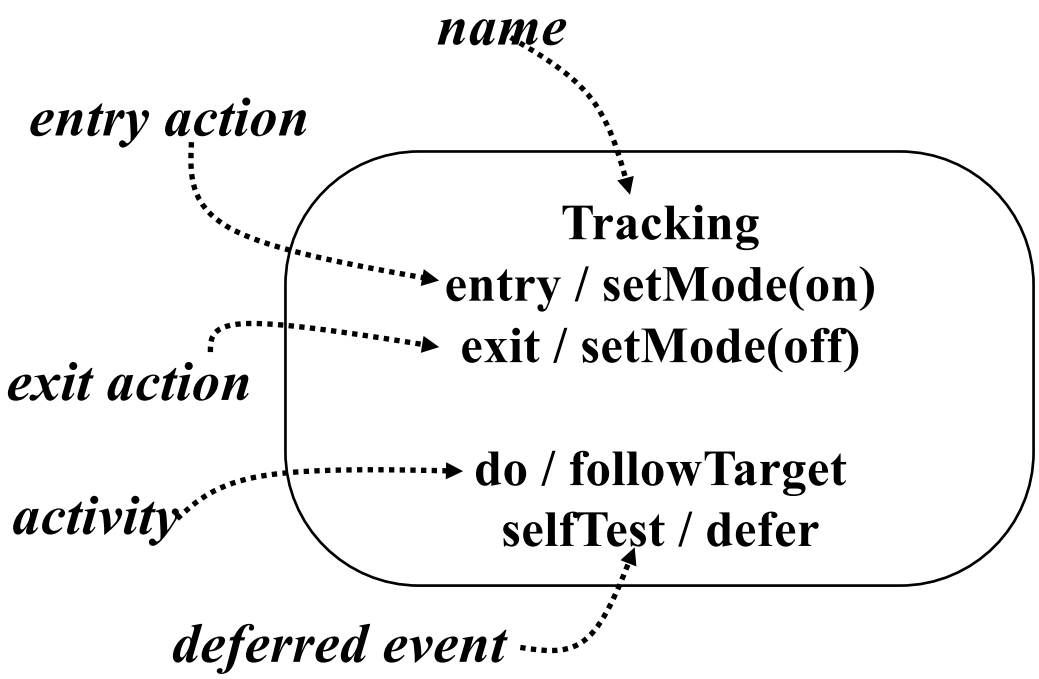
- UML sequence diagram represent *behavior in terms of interactions*
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent structure).

UML Statechart diagram

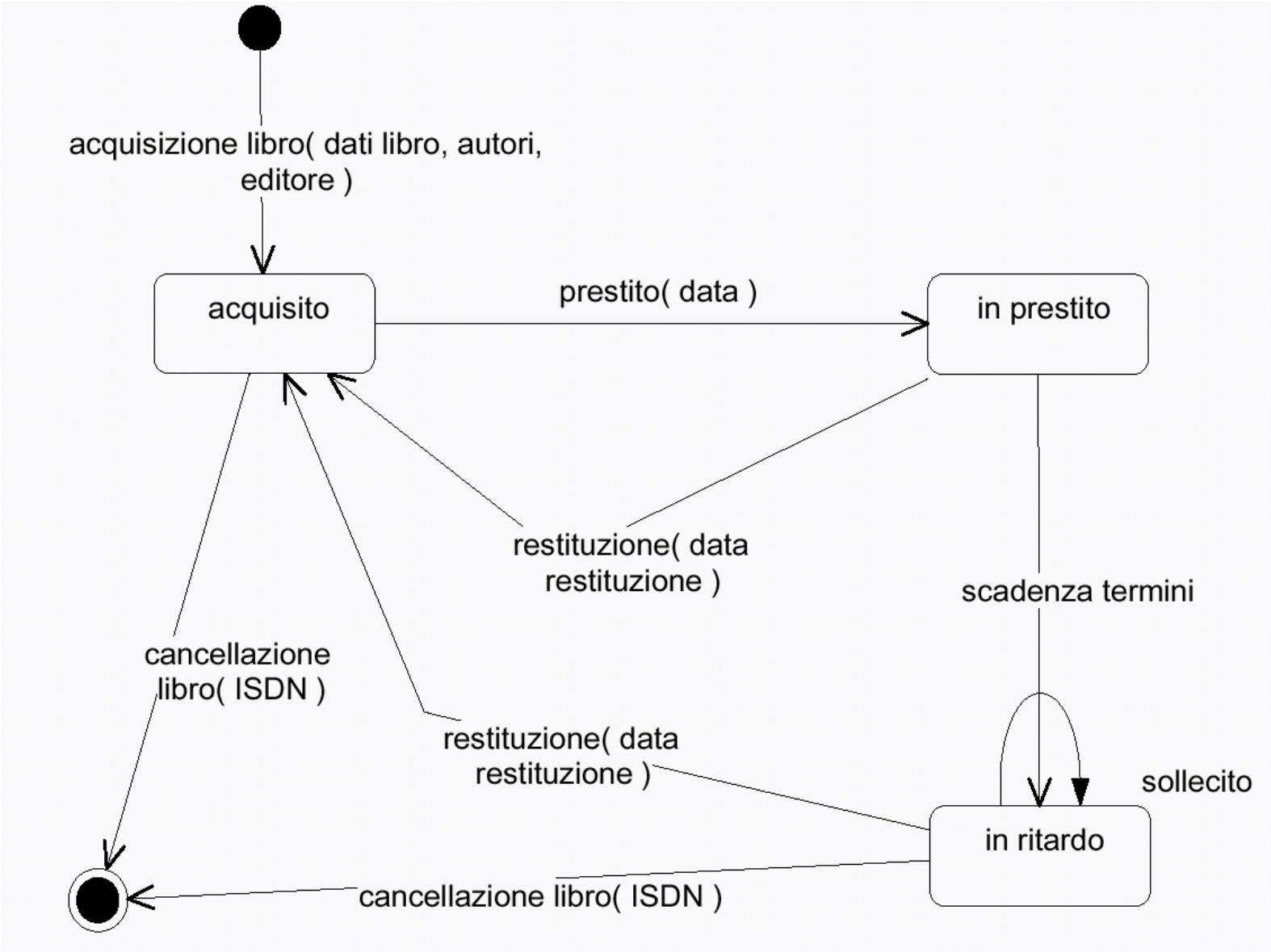


Represents behavior of *a single object* with interesting dynamic behavior.

Stato completo



Esempio: Libro



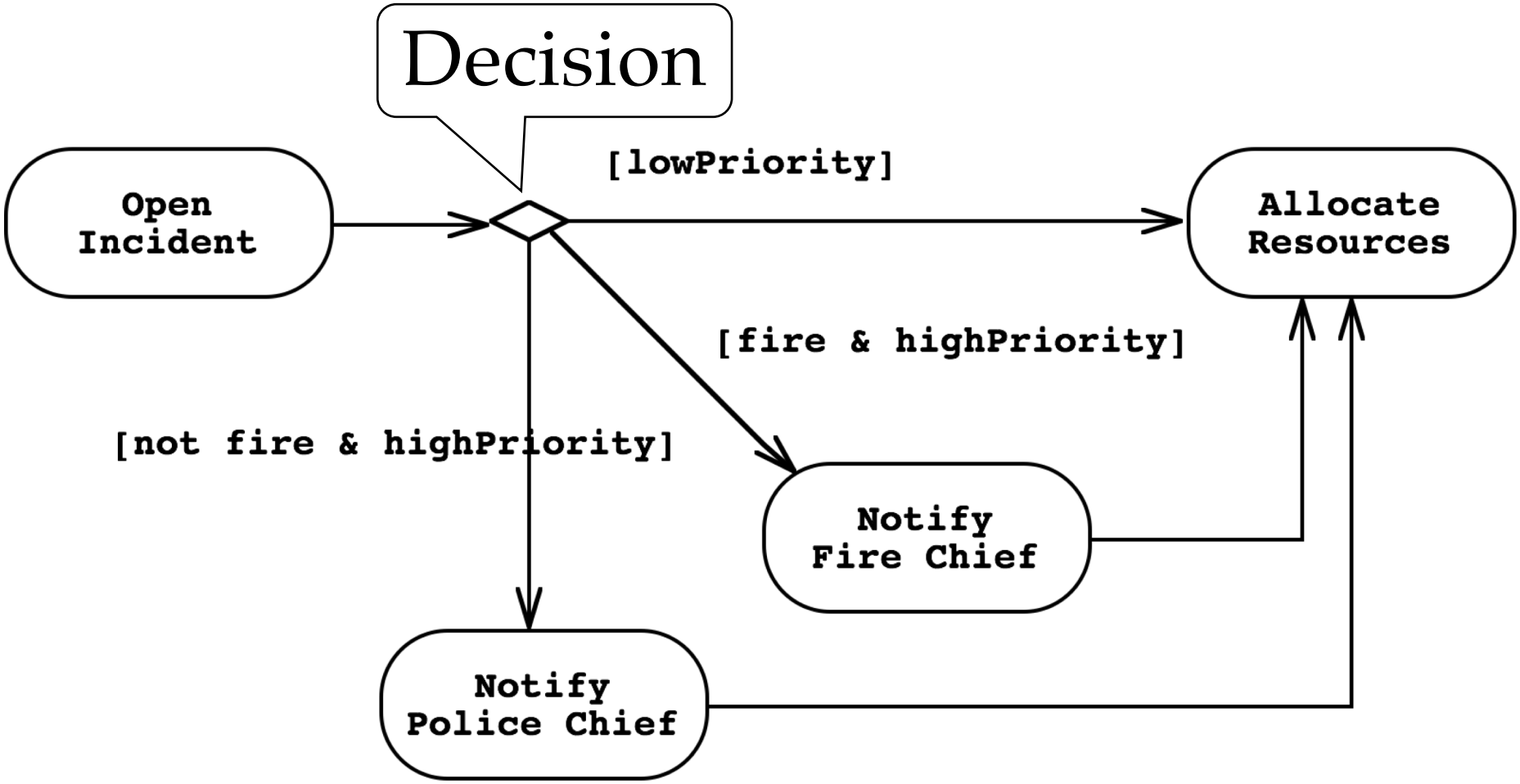
Activity Diagrams

- An activity diagram is a special case of a state chart diagram
- The states are activities (“functions”)
- An activity diagram is useful to depict the workflow in a system

UML Activity Diagrams

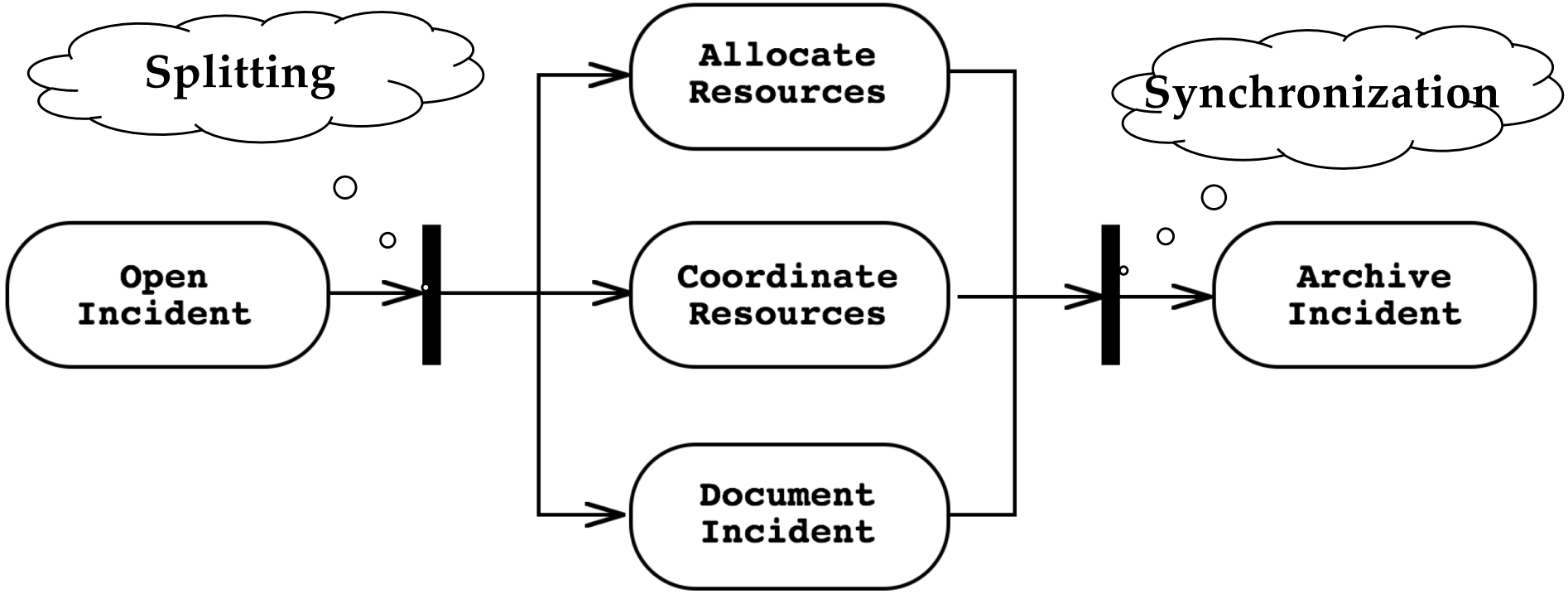
- An activity diagram is a special case of a state chart diagram
- The states are activities ("functions")
- An activity diagram is useful to depict the workflow in a system.

Activity Diagrams allow to model Decisions



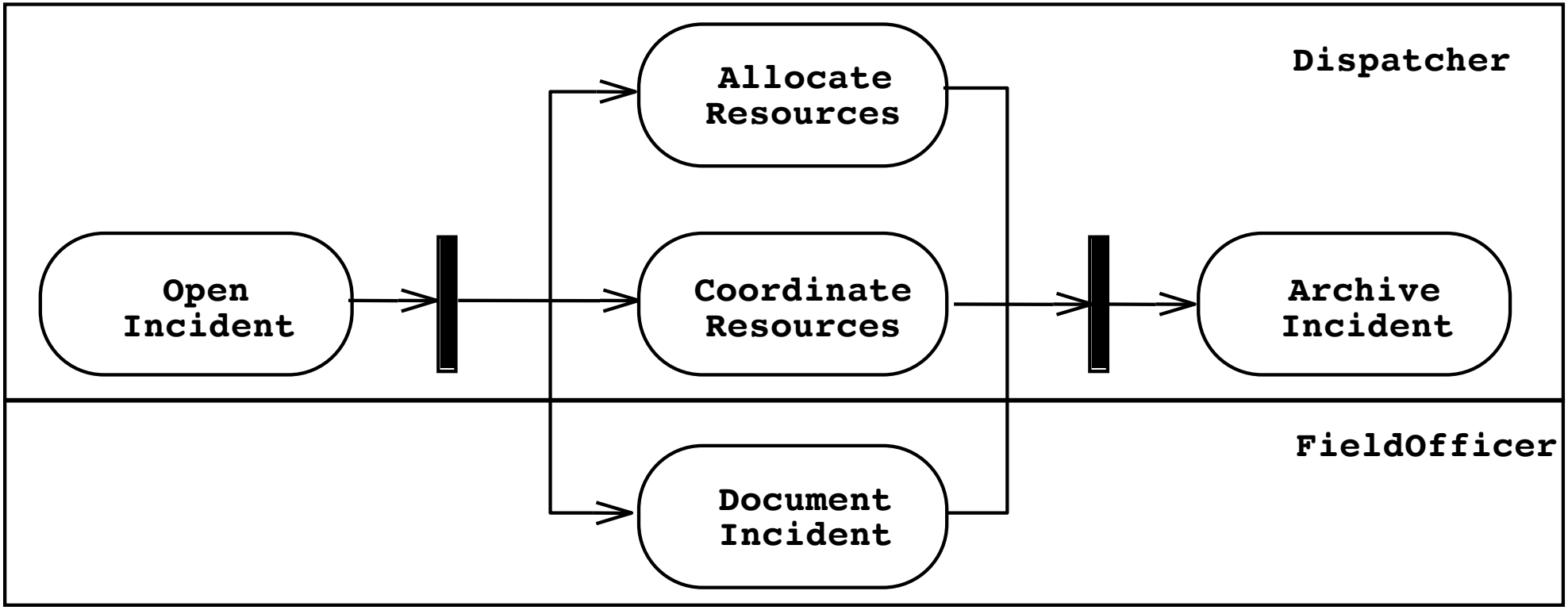
Activity Diagrams can model Concurrency

- Synchronization of multiple activities
- Splitting the flow of control into multiple threads

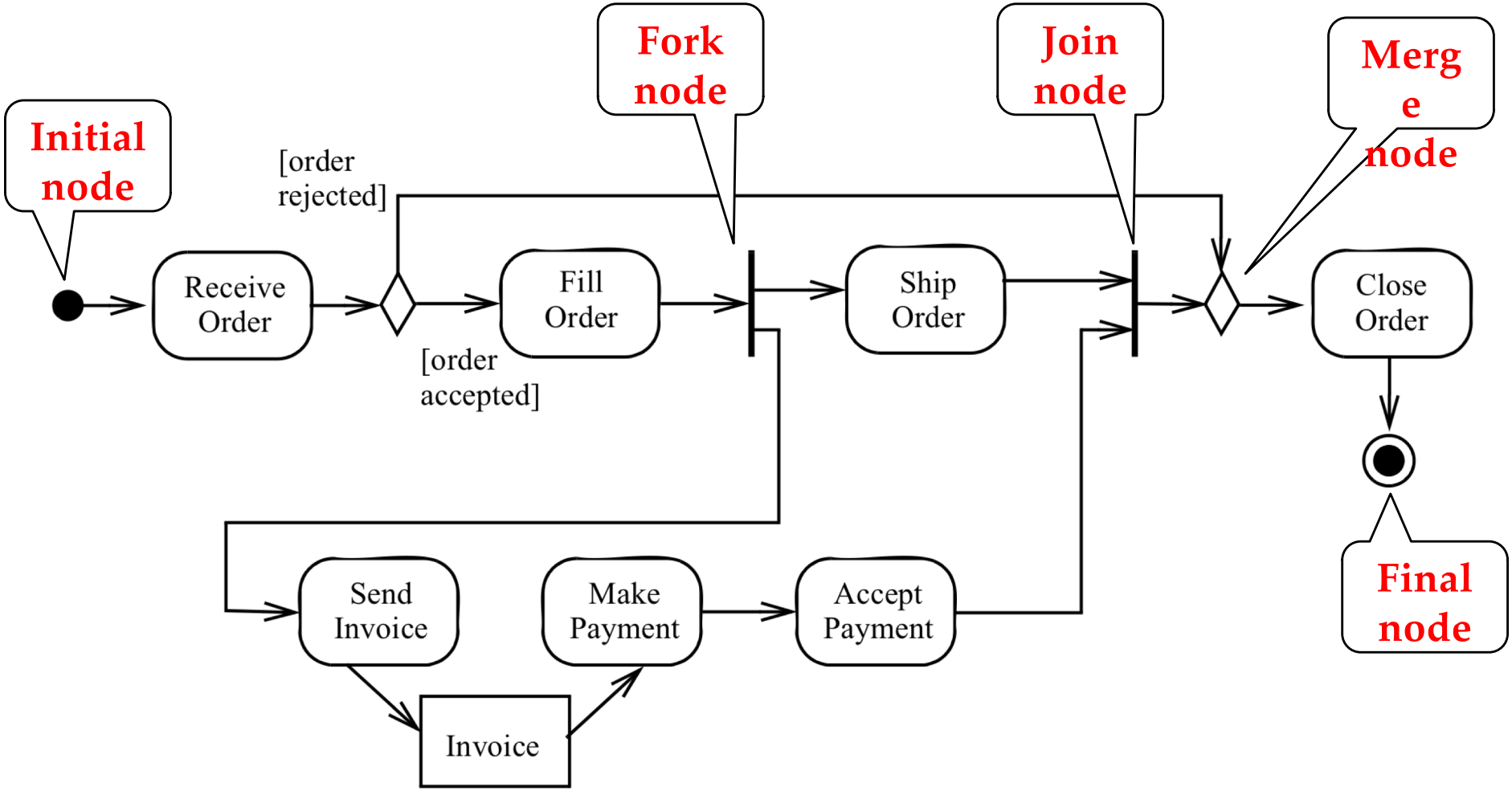


Activity Diagrams: Grouping of Activities

- Activities may be grouped into swimlanes to denote the object or subsystem that implements the activities.



Activity Diagram Example



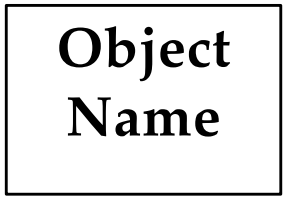
Activity Diagram: Activity Nodes & Edges

- An activity diagram consists of nodes and edges
- There are **three** types of activity nodes
  - ✓ Control nodes
  - Executable nodes
    - Most prominent: **Action**
  - Object nodes
    - E.g. a document
- An **edge** is a directed connection between nodes
  - There are two types of edges
    - Control flow edges
    - Object flow edges

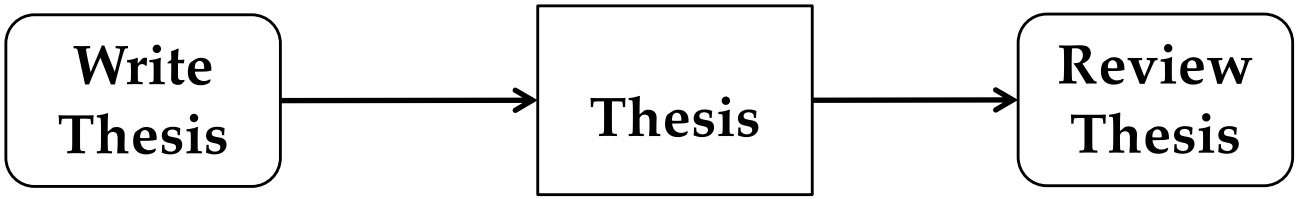
Action Nodes and Object Nodes

- Action Node

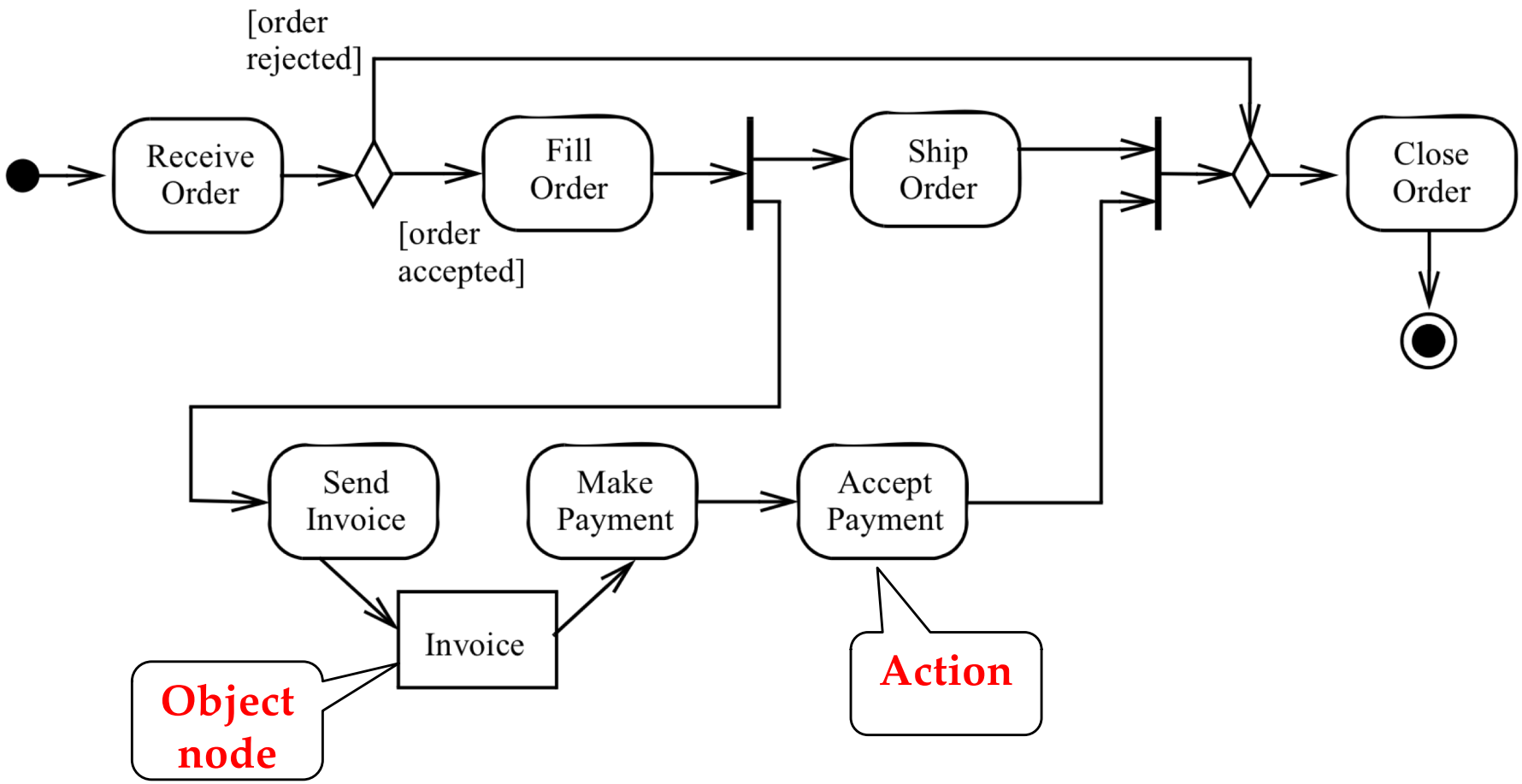
- Object Node



- An action is part of an activity which has local pre- and post conditions
- Historical Remark:
  - In UML 1 an action was the operation on the transition of a state machine.



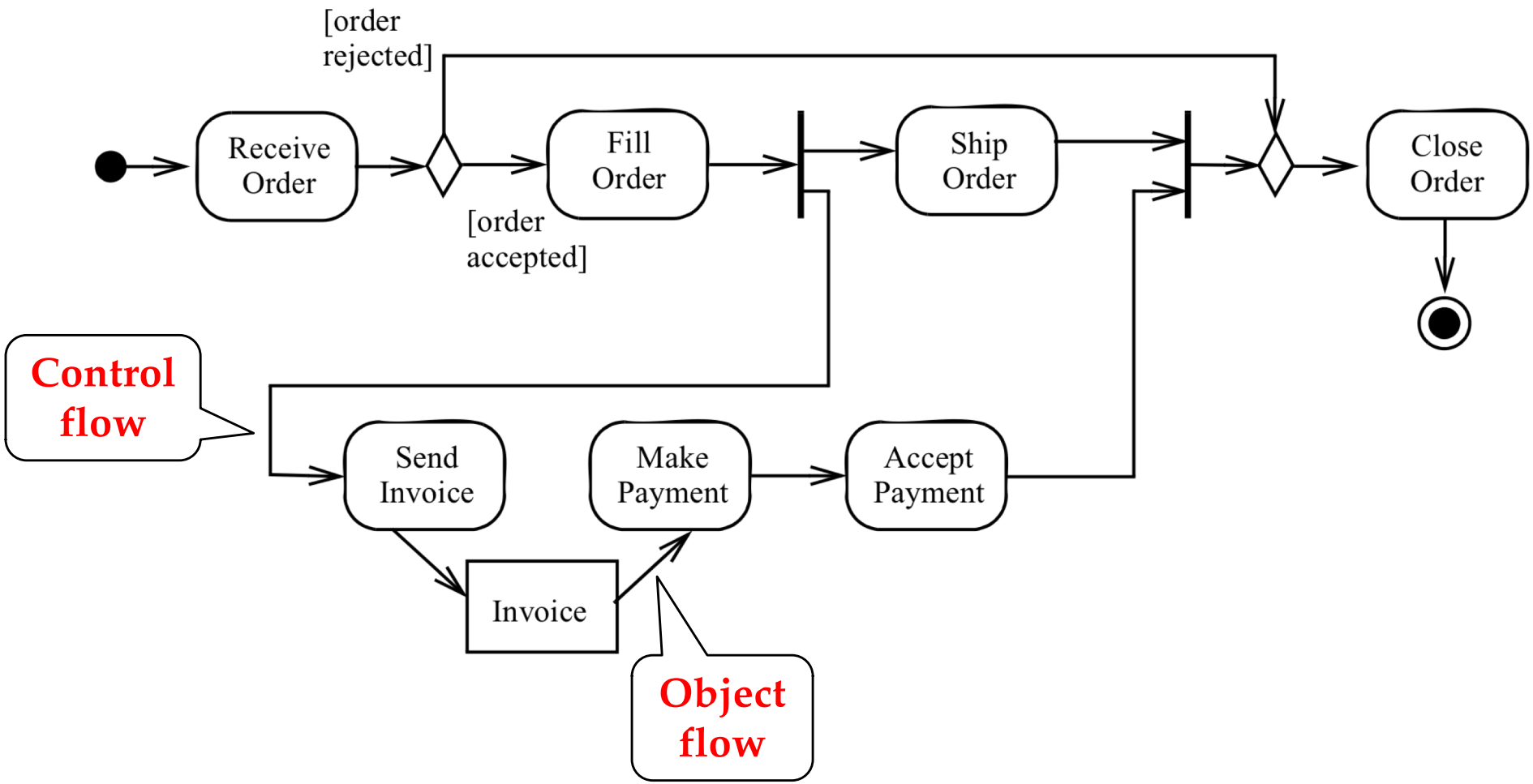
Activity Diagram Example



Activity Diagram: Activity Nodes & Edges

- An activity diagram consists of nodes and edges
- There are **three** types of activity nodes
  - ✓ Control nodes
  - ✓ Executable nodes
    - Most prominent: **Action**
  - ✓ Object nodes
    - E.g. a document
- An **edge** is a directed connection between nodes
  - ➡ There are two types of edges
    - Control flow edges
    - Object flow edges

Activity Diagram Example



Summary: Activity Diagram Example